October 1983

Designing with the System Friendly 2817A 5V-Only E^2PROM

Justin Orion
Applications Engineer
Intel Corporation

©INTEL CORPORATION, 1983.

ORDER NUMBER: 230714-001

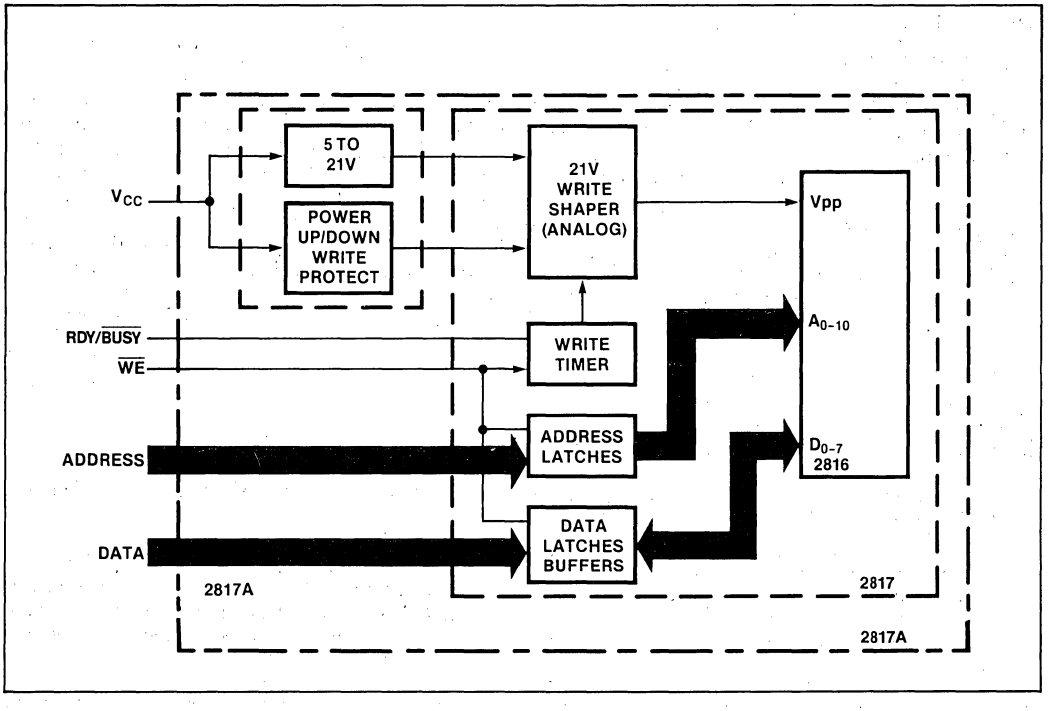


Figure 1. E²PROM Evolution: Increasing Intelligence On-Chip

Advantages of an Intelligent E²PROM

The 2817A has on-chip latches and timing which allow it to connect directly to the microprocessor data bus. Writing a byte to the 2817A is accomplished by sending a write pulse to the part. The timing for this pulse is the same as the timing for a static RAM write cycle. Upon receiving this write pulse the 2817A starts a write operation. During a write operation the 2817A's data lines go to a high impedance state, allowing normal processing to continue on the microprocessor data bus. After the write operation has been completed, the system CPU is notified via the 2817A's RDY/$\overline{\text{BUSY}}$ output. The RDY/$\overline{\text{BUSY}}$ signal frees the designer from having to time the E²PROM write operation by either hardware timing logic or software time-out loops.

Note that while performing a write operation the 2817A will not respond to read or write requests. The 2817A's data bus will remain in a high impedance state, regardless of the input control signals, until the write operation has been completed.

APPLICATIONS FOR E²PROMS

Application Area #1: Firmware Remote Downloading

Remote downloading is important in any system that uses firmware to store program code. Storing software in ultraviolet erasable EPROMs has the advantages of being non-volatile and allowing zero wait-state execution with high-speed processors such as the 8 MHZ 8086-2. A limitation of this method of firmware storage is that system software usually has bugs in the first year or so and is frequently improved or upgraded. To make a firmware change in a system with UV EPROMs a serviceperson must go to each customers site, partially disassemble each system, replace the EPROMs, and re-assemble the system. Now, by storing some or all of the system software code in E²PROMs, software changes can be made by telephone, without sending a serviceperson to the customer site. To make software changes in E²PROMs, the serviceperson calls the customer site, connects the system to the service center computer via telephone, and transfers the new code to the customer's system. In less than a minute the system can be up and running with the new software at a minimal cost to the manufacturer or customer.

Firmware stored in E²PROM can be changed/upgraded remotely over any type of communications link.

All E²PROM or a Combination of EPROM/E²PROM?

System firmware can be completely stored in E²PROM to allow remote alterability of all code, or a combination of E²PROM and UV EPROM can be used.

If the firmware is completely stored in E²PROM, any or all of the firmware can be replaced electrically. This type of system is shown in Figure 2. When the system is prepared for shipping, the operating system software is downloaded directly into the E²PROM memory after the E²PROMs have been installed in the system. When the software is changed, upgraded, or expanded the entire new software package can be loaded remotely over the telephone.

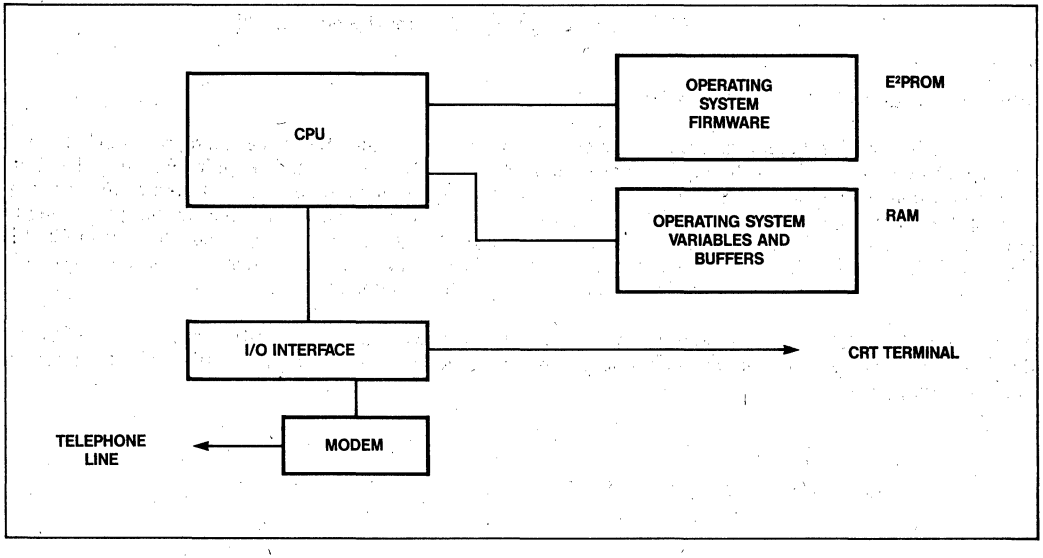


Figure 2. Total Firmware Storage in E²PROM

In a system with a large amount of program code, it may be desirable to use EPROM for storing most of the firmware and use E²PROM for the corrected portions of the code. (see Figure 3) This method, sometimes called "firmware patching," allows a system to use high density EPROM to store a firmware package that is segmented into a number of sections or routines. A small part of the system's E²PROM is used to store a short program that calls each routine as it is needed (see Figure 4). When an error is found in one of the firmware routines, a corrected version is loaded into the extra E²PROM space. The new corrected routine is then called instead of the erroneous routine by changing the calling program (see Figure 5). The corrected routine and new call command can be sent directly from a service center to the system at the customer site over any telephone line.

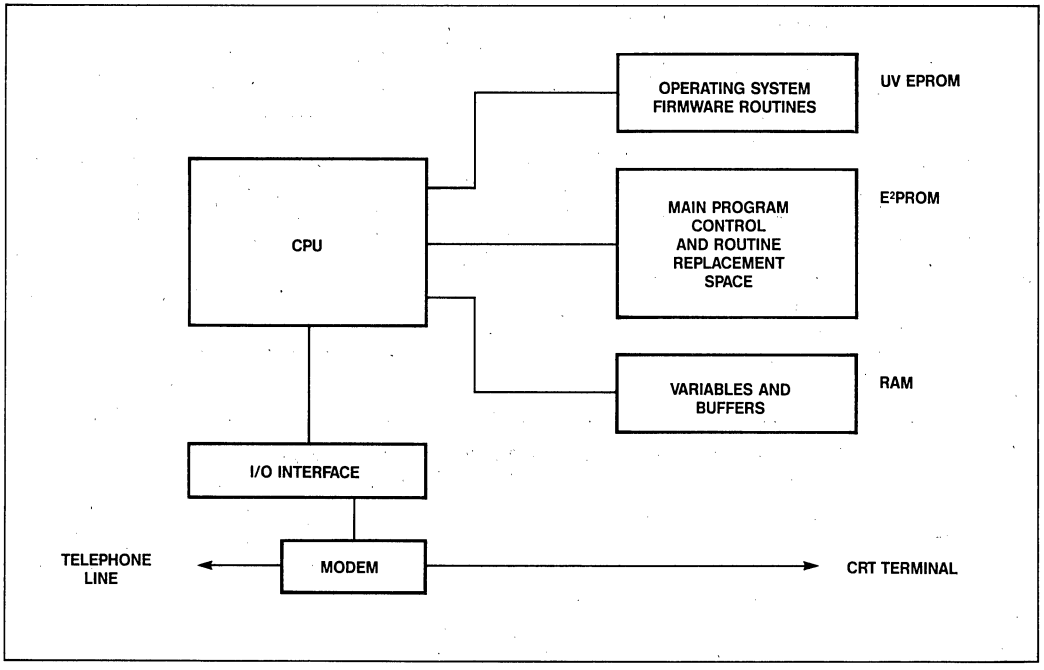


Figure 3. Partial Firmware Storage in E²PROM

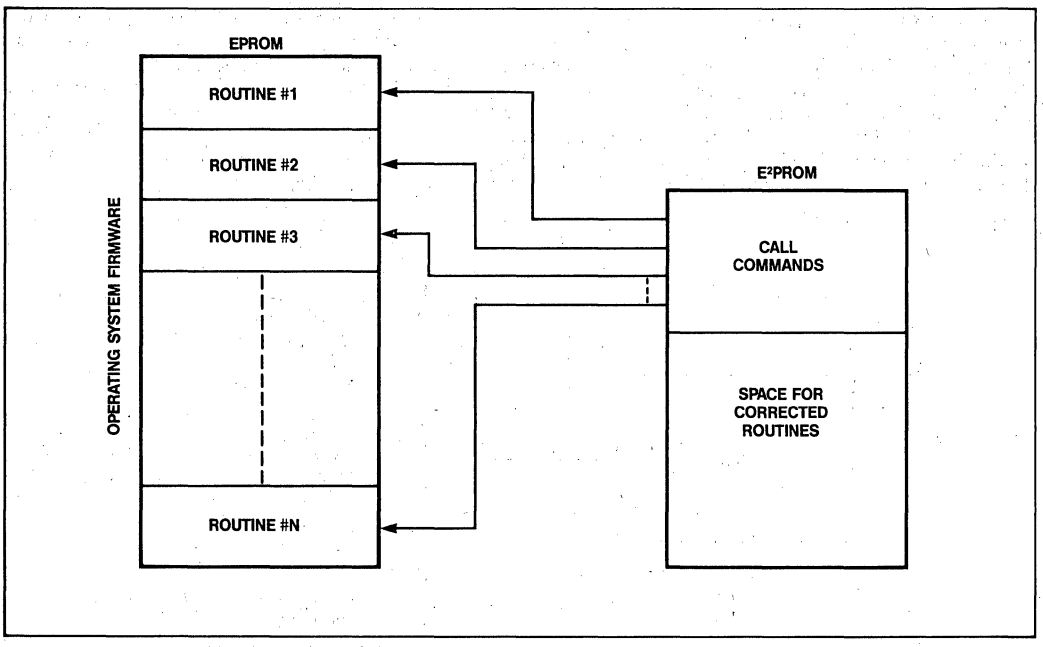


Figure 4. Firmware Patching: Operating System Memory Map

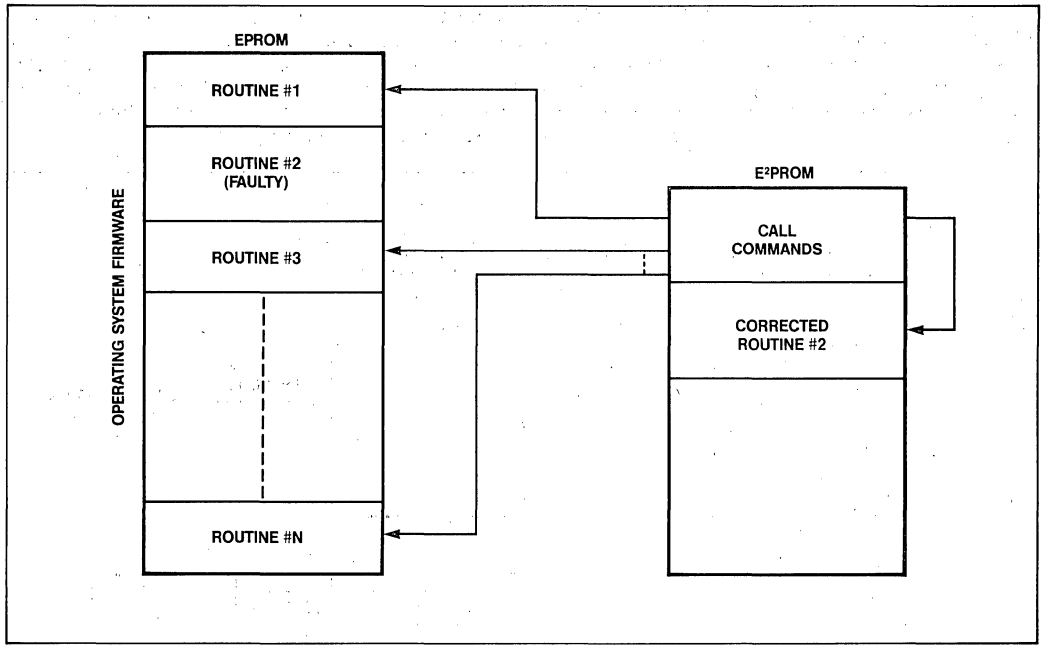


Figure 5. Firmware Patching Example: Replacement of Faulty Routine with Corrected Routine in E²PROM

Easy Software Upgrades

Software corrections, upgrades, or the addition of software options are easily done in a system using E²PROMs for complete or partial firmware storage. The following is a typical example.

The OEM's service center calls the customer on the telephone and tells him that the center wishes to upgrade the customer's system firmware. The center instructs the system operator to set a "download" switch on the system, connect the phone to the system, and depress the system RESET switch. The new software is transferred from the center to the customer system, where it is stored in E²PROM. The new software package then tests itself and the system. The system's CRT display keeps the operator informed of each step in the process. When the system test is done, the CRT display instructs the operator to disconnect the phone, turn off the "download" switch, and reset the system. The system is now up and running with the new software after only 40 seconds of down time. (see section on Fast Array Programming on page 29.)

Hardware Implementation of Remote Downloading

As the new code is being received over the phone from the service center it is loaded into RAM. The code is sent in blocks and a checksum is sent with each block. If a data error occurs due to a poor phone connection, the block is re-transmitted. Once the entire program has been correctly received, it is transferred to E²PROM.

In a system using both EPROM/ROM and E²PROM the downloading software routine is executed out of the EPROM/ROM. In a system with only E²PROM, the downloading routine can be executed either out of the system RAM or out of E²PROM. With the first method the downloading routine is transferred from E²PROM to RAM. (see Figure 6) The routine is then executed out of RAM and the new code is loaded remotely over a communications link into E²PROM. A copy of the downloading routine is included in the new code in the E²PROM for future remote upgrades. At the end of the transfer operation, control is passed back to the new program in E²PROM.

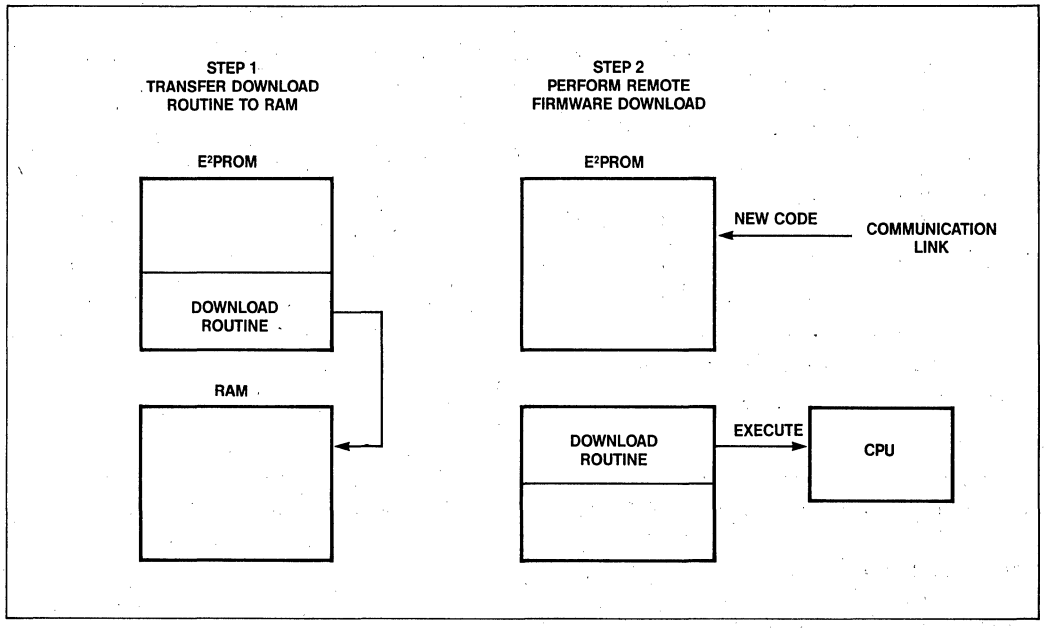


Figure 6. Remote Downloading to an E²PROM-only System with Download Routine in RAM

In systems where there are two or more E²PROMs the downloading routine can be initially executed out of the first E²PROM. (see Figure 7) After the new program code has been loaded into the rest of the E²PROMs in the system, execution control is passed to a copy of the download routine in E²PROM #2, as shown in Figure 7. The remaining new code is then loaded into E²PROM #1.

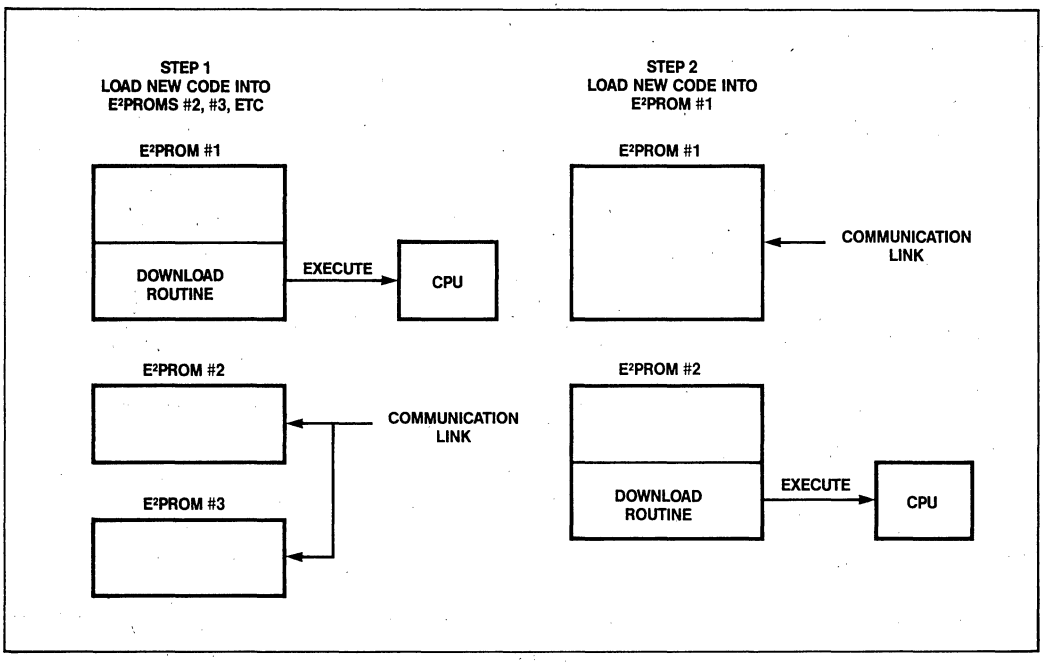


Figure 7. Remote Downloading to an E²PROM-only System with Downloading Routine in E²PROM

Demo Circuit

The 2817A/8088 stand-alone system design shown in this applications note can send data over a serial RS232 cable, as shown in the block diagram in Figure 8. This design can also send over any telephone line using an acoustic coupler. The hookup for transmission over a phone line is shown in Figure 9. The software on the applications demo board is set up to send text messages between boards when they are connected as shown in Figure 8 or 9. The messages are then stored in E²PROM memory where they remain unchanged until it is desired to delete a message or clear the message array. It can be seen that program code rather than text messages can just as easily be transferred between the two systems and stored in E²PROM memory. The downloading routine would be transferred to and executed from the system RAM. The new program code would then be executed directly out of E²PROM.

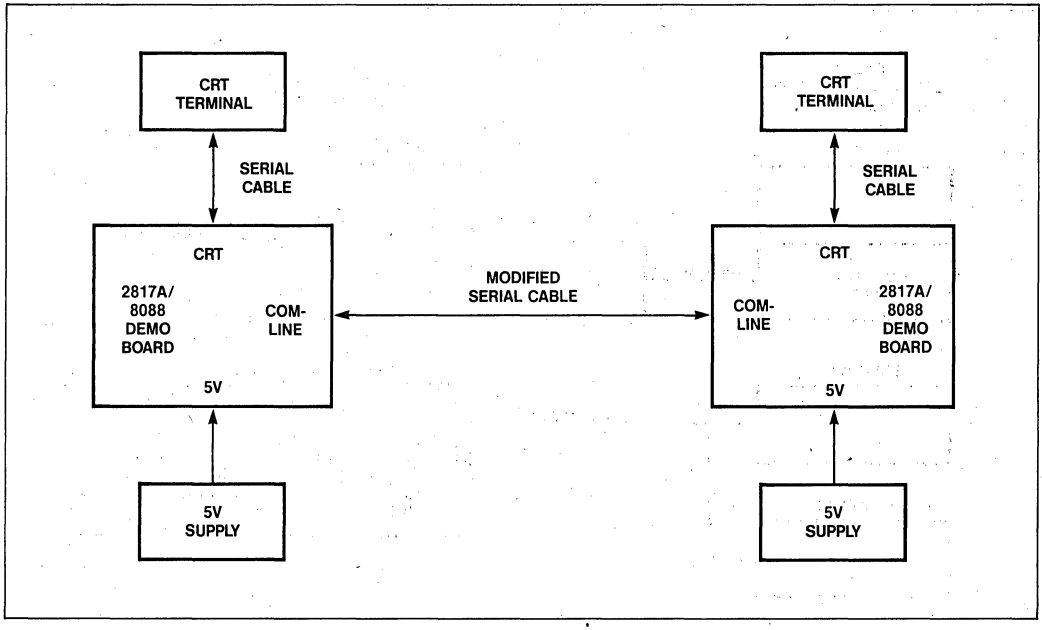


Figure 8. Remote Download Demo Hookup

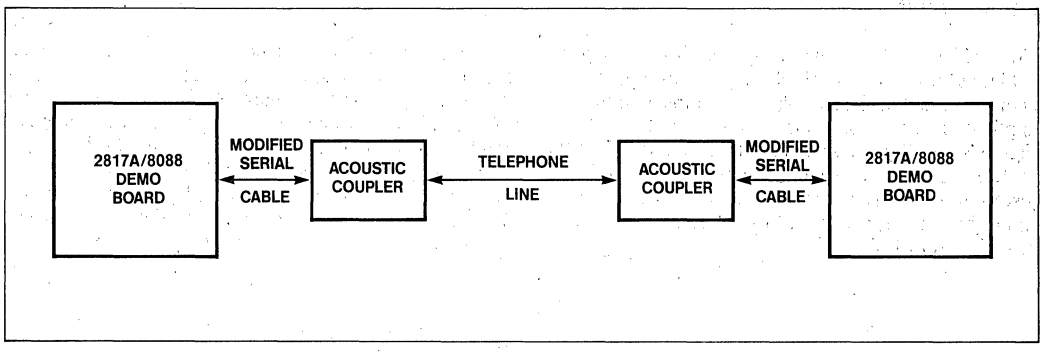


Figure 9. Remote Downloading over a Telephone Line using the 2817A/8088 Demo Board

The "modified serial cable" shown in Figure 8 is an RS232 serial cable with the send and receive lines switched as shown in Figure 10.

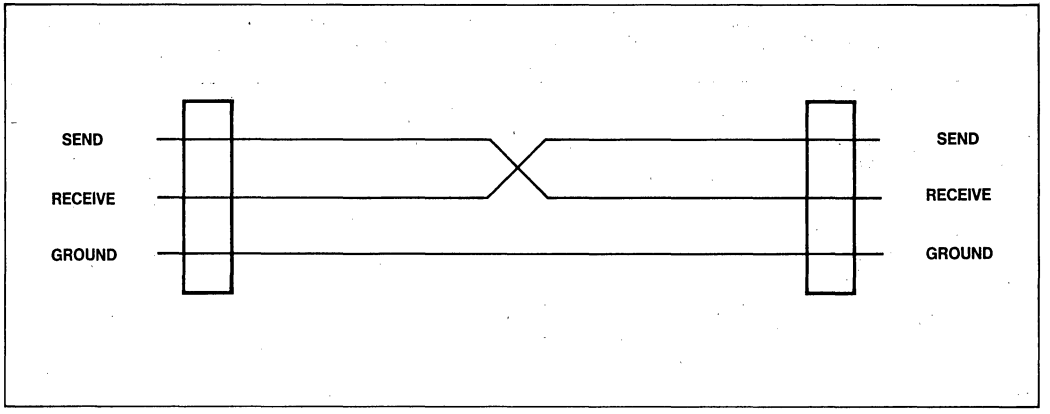


Figure 10. Modified RS232 Serial Cable

The 8088 software for performing a data block transfer between systems is shown on pages 42-44 in Appendix A. Figures 11 and 12 show the flow charts for the data transfer software routines.

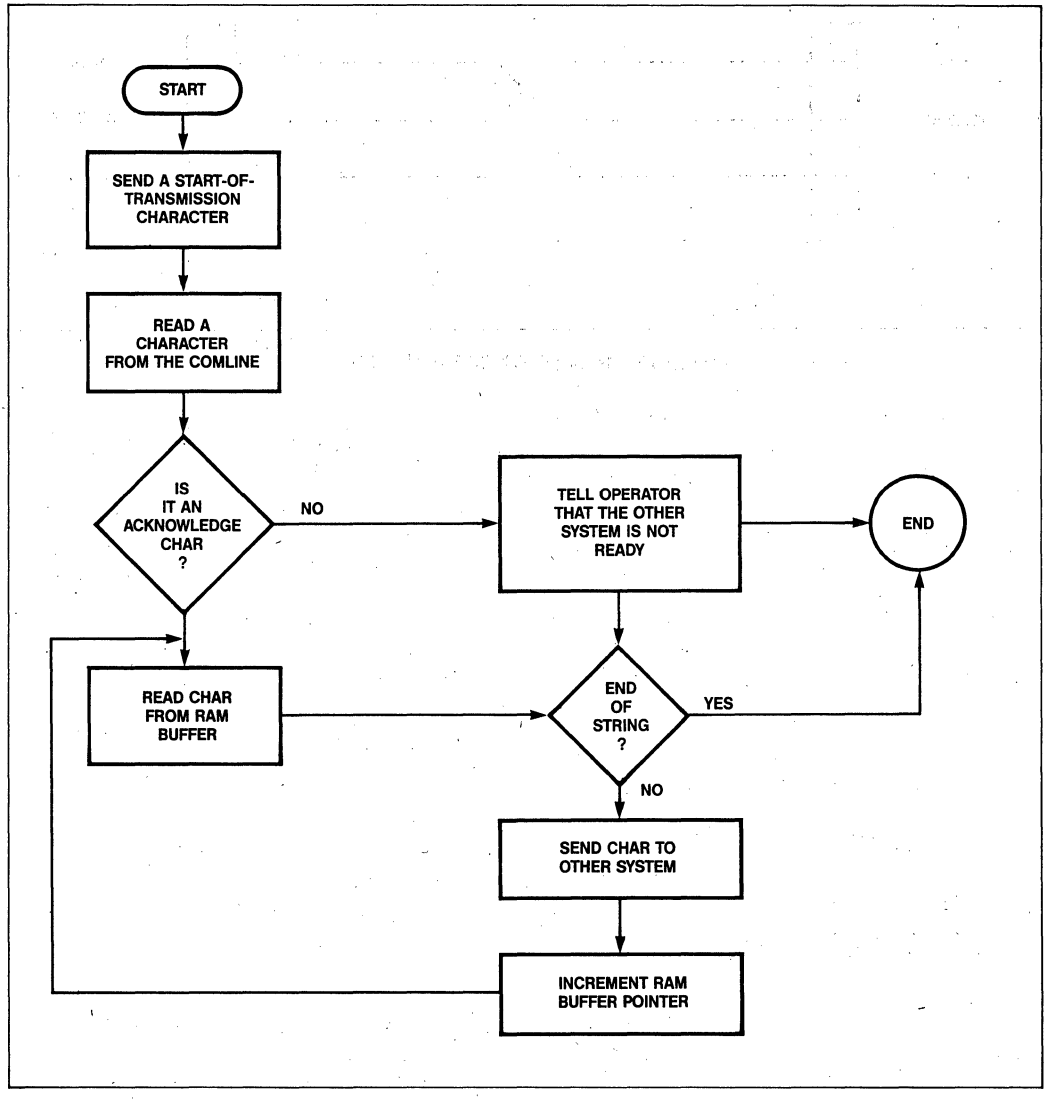


Figure 11. Data Transfer Software: Transmitting System

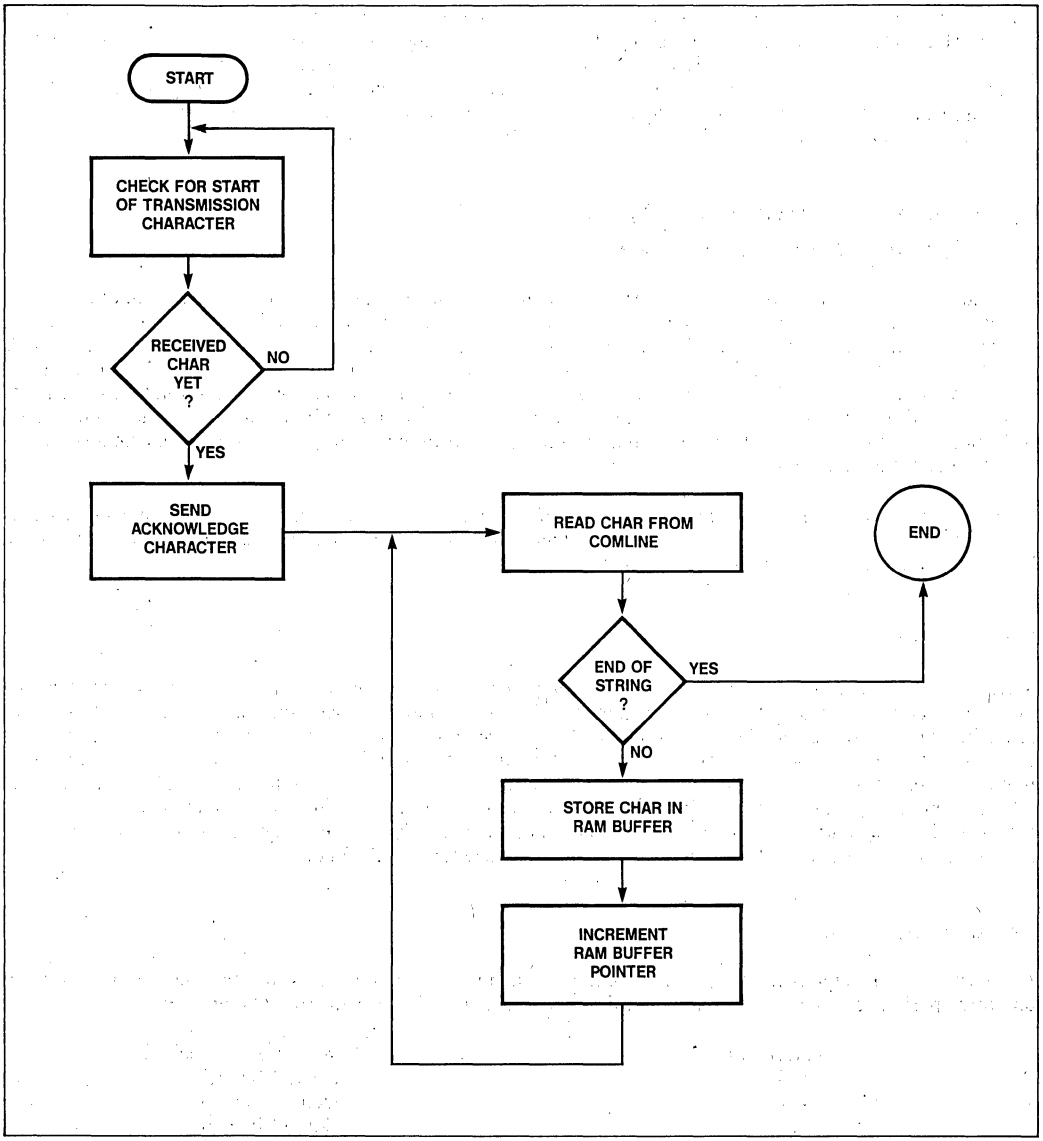


Figure 12. Data Transfer Software: Receiving System

Since serial channel drivers in this design do not provide full spec RS232 levels, the serial cables should be kept under 10 feet in length. The drivers are less than full spec to reduce the complexity of this system for demonstration purposes. These drivers require only a 5V supply to operate rather than the +12V and −12V required for standard RS232 interfaces. The circuit description for the RS232 serial channel drivers under the section, "2817A/8088 Applications Demo Board Circuit Operation"gives more information on the circuitry and also shows a design example for a full spec RS232 interface.

Application Area #2: System Reconfiguration

Small Systems—User Friendly Operation

In small systems such as modems, point of sale terminals, and data terminals, hardware switches have been used to set operation parameters such as baud rates, synchronous/asynchronous selection, and data constants (tax rates, display attributes, and many others). It is usually necessary to have printed tables and operation manuals in order to understand how and when to set these switches. Wouldn't it be much easier if the CRT terminal asked you what baud rate you wanted or drew a pricing table on the screen and asked you to fill it in? With E²PROM this is easily done. Parameters entered by the operator can be stored in specific blocks, called "look-up tables," in E²PROM. Each time the system is powered up, the CPU sets all parameters based on the information stored in the E²PROM look-up table.

E²PROM may also be used to re-define the function of any given key of a terminal keyboard. Intelligent terminals and graphics terminals with these "soft keys" are highly flexible in that each user can define the functions he needs for the most efficient use of the terminal.

Large Systems

Medium and large sized computer systems and computer system networks often have hardware switches and jumpers to control peripheral channel assignments, data rates, and even user accessibility. E²PROMs are now replacing these switches and allowing the parameters or assignments to be changed via software. Special access codes in firmware allow only restricted access to critical function controls. There is no need to have manuals on hand to make simple peripheral channel changes. The firmware or software displays a "menu" on the terminal CRT screen and asks the user to fill in the desired numbers or parameter values. These numbers and values are then stored permanently in E²PROM look-up tables, and the system's parameters and operational modes are set according to the values stored in E²PROM each time the system is powered up.

Demo Board Circuit

For the purpose of demonstrating system re-configuration, imagine a computer room with three computer systems and three peripherals. The systems are identified as #1, #2, and #3. The three peripherals are:

Peripheral	Indicated on demo board as:
Hard Disk	Demo Channel A
Printer	Demo Channel B
Mag Tape Drive	Demo Channel C

A peripheral control unit is used to determine which system is connected to which peripheral, as shown in Figure 13. More than one peripheral can be allocated to a given system. An E²PROM memory in the peripheral control unit acts as a software switch to permanently retain the present peripheral configuration until changed by an operator. As the data storage requirements of each computer system changes, the various peripherals can be re-assigned as needed. For more efficient operation a routine could be written which would automatically re-configure the peripheral channels based on the data storage requirements/requests of the systems. The peripheral assignments on the 2817A/8088 demo board are changed by a software command from the operator. The new assignments are stored in a look-up table in E²PROM as shown in Figure 14. Figure 15 shows the location of the look-up table in E²PROM. In this manner the present system configuration remains in effect through system power-down and power-up.

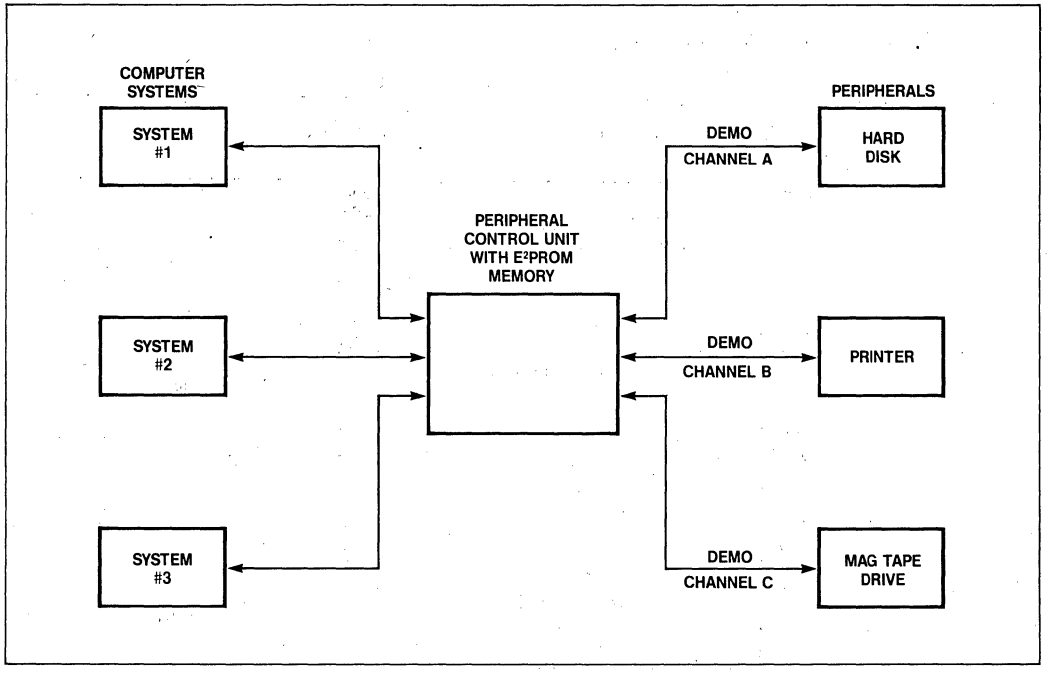


Figure 13. System Re-configuration Demo: Computer Room

		PERIPHERAL
BYTE #1	SYSTEM #	(DEMO CHANNEL A) HARD DISK
BYTE #2	SYSTEM #	(DEMO CHANNEL B) PRINTER
BYTE #3	SYSTEM #	(DEMO CHANNEL C) MAG TAPE DRIVE

The value "System #" Determines Which System has access to the Peripheral.

Figure 14. System Re-configuration Look-up Table

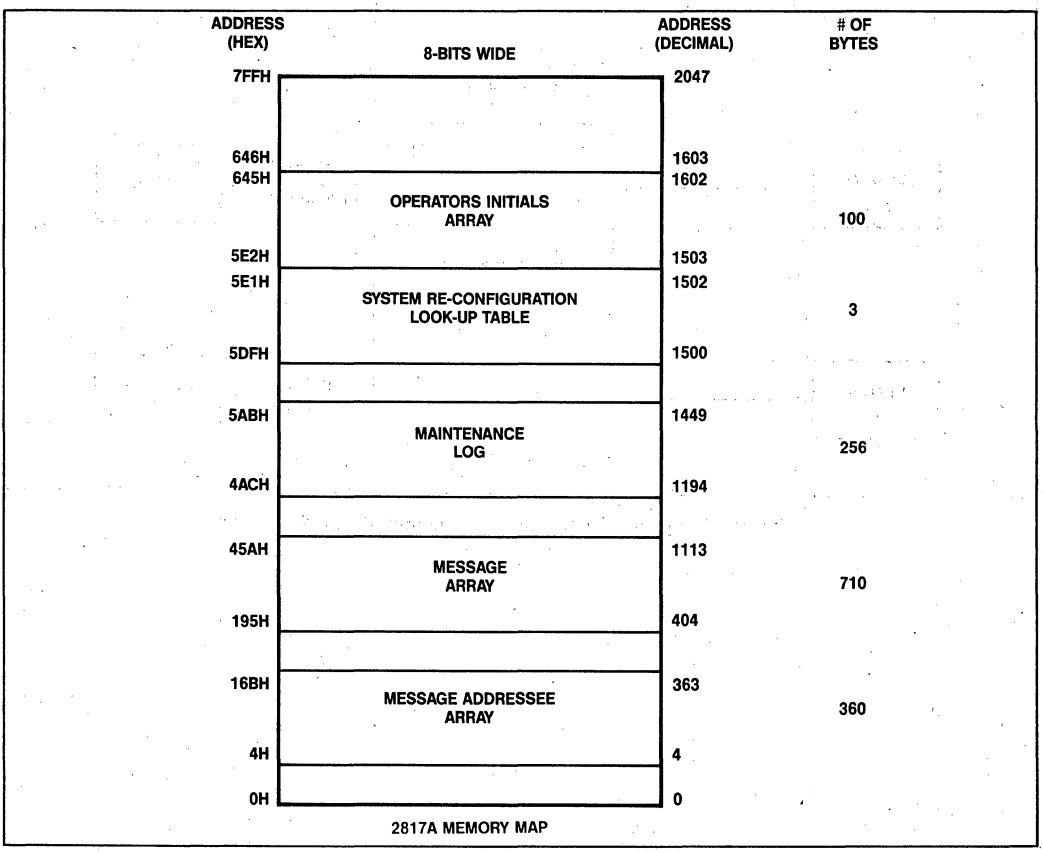


Figure 15. 2817A/8088 Applications Demo

Further detailed information on the hardware operation of the System Reconfiguration Demo is given in the section "2817A/8088 Applications Demo Board Circuit Operation" under "Demo Channel Display LED Ports."

Application Area #3: Maintenance Log

When equipment is repaired or upgraded it is important to record the nature of each repair problem and to document the revision levels. The information, usually filed at the user's site, is often lost. E²PROM can be used to store maintenance record information and revision level numbers in the system itself, thus eliminating possible confusion with other similiar systems. With an E²PROM maintenance log, the information is always readily accessible through the software. A few simple software commands can be used to enter or display maintenance log information. Complete information about the system's optional equipment and even the serial number can be stored in the system's E²PROM log. This information may also be read over a telephone line by the manufacturer's service center computer, eliminating the possibility of operator error in relaying the information.

Demo Board Circuit

The demo board has an E²PROM maintenance log. The memory space allocated for this log in the 2817A is shown in Figure 15. Entries are made from the keyboard, and the contents of the log can be displayed on the CRT at any time. The entries are automatically numbered as they are entered.

Application Area #4: Electronic Message Storage

E²PROM can be used to store messages in a small stand-alone system. The low cost and ease of use of the 2817A makes it very desirable for small microprocessor-based designs. Visualize one of the numerical control systems that supervises the various phases of production in an industrial plant. To keep shift operators informed of any process changes, messages can be displayed on the CRT display. These messages, stored in the system's E²PROM by a previous operator or a supervisor, inform future operators of temporary or permanent operation modifications, or warn them of any problems that should be closely monitored. No floppy disk needs to be left in the terminal to store the message: all that is needed is the in-system E²PROM. This kind of message storage capability is also a useful feature on personal computers, personal development systems, and intelligent modems.

Another application for E²PROM message storage is re-programmable restricted access keys to computer systems. In a system using E²PROM for security control, assess to critical data can be restricted based on specific text strings, or "keys", that are entered by the operator. In this type of system the "keys" are stored in E²PROM and are changed as needed by an authorized company officer. For example, a data processing system using this technique allows access to sensitive product cost tables only upon receipt of a specific text string that is determined by an authorized company officer. The text string can be as simple as "code table" or can be related to a product line name such as "Intel Personal Development System". The text key can also be a completely unrelated phrase such as "Donald Duck" or anything else that is easy for an authorized operator to remember, yet prevents general access. Such a security system is highly reliable because it does not depend upon electro-mechanical storage device—it uses reliable, easy-to-design-in E²PROMs.

Demo Board Circuit

The demo board can accept a message destined for a given addressee. The message and the name are kept in separate arrrays as shown in Figure 15. Whenever the board is powered up, the names in the addressee array are listed on the CRT, thus giving notice of the names of the people who have messages. The messages for any specific addressee can be viewed on command. Messages for any specific addressee can later be deleted, or the entire message array can be cleared. The list of addressees can also be viewed at any time on command.

MICROPROCESSOR INTERFACING

General Concepts

The 2817A is designed to be easily interfaced to a microprocessor. The control signals, $\overline{CE}$, $\overline{OE}$, and $\overline{WE}$ allow the simple basic design shown in Figure 16 to be used. By using the two-line control concept incorporated in all Intel 28-pin memories, the possibility of bus contention with other memory devices on the same data bus is eliminated. This is accomplished by connecting the $\overline{CE}$ input to the output of the system memory address decoder and by connecting the CPU's read and write command signals to the $\overline{OE}$ and $\overline{WE}$ inputs, respectively, as shown in Figure 16. In single line control the data bus is driven as a function of the addresses. This causes overlap on the $\overline{CE}$ select lines which can result in data bus contention as shown in Figure 17. Through two-line control, the data bus is driven only when both the address select ($\overline{CE}$) and the memory command ($\overline{RD}$) are active (see Figure 18).

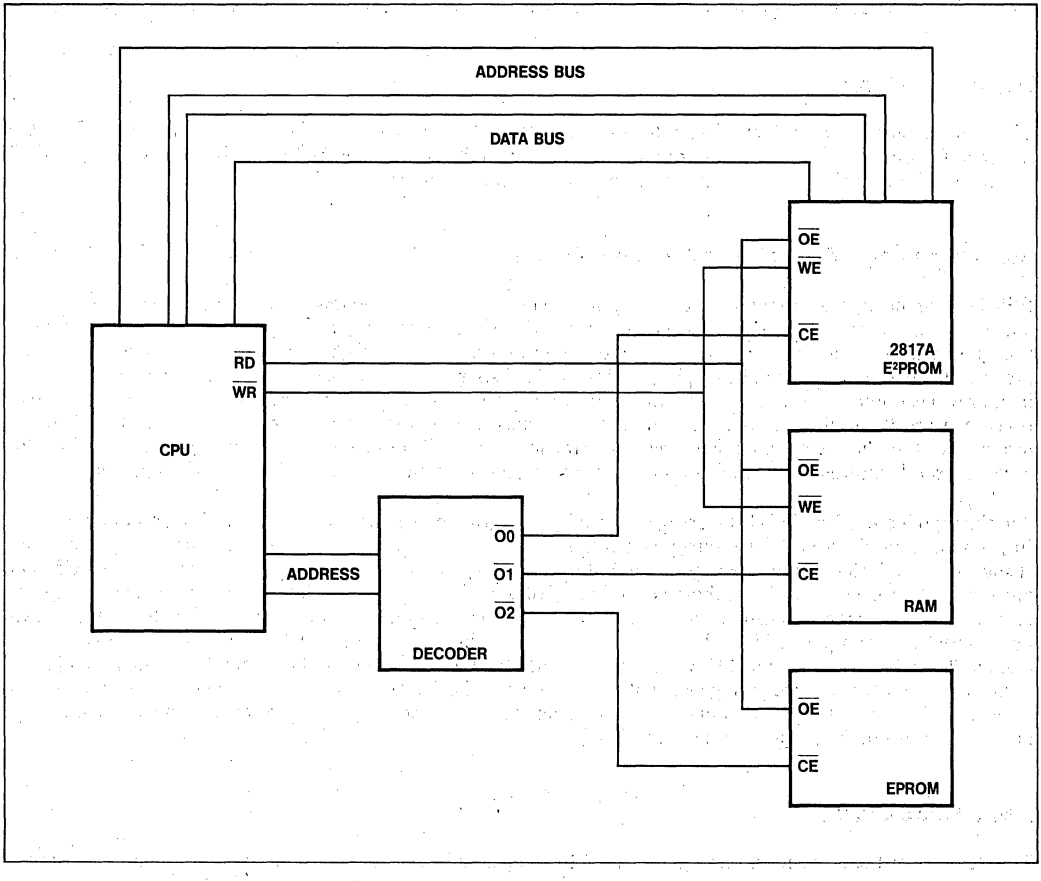


Figure 16. Basic Microprocessor Interface

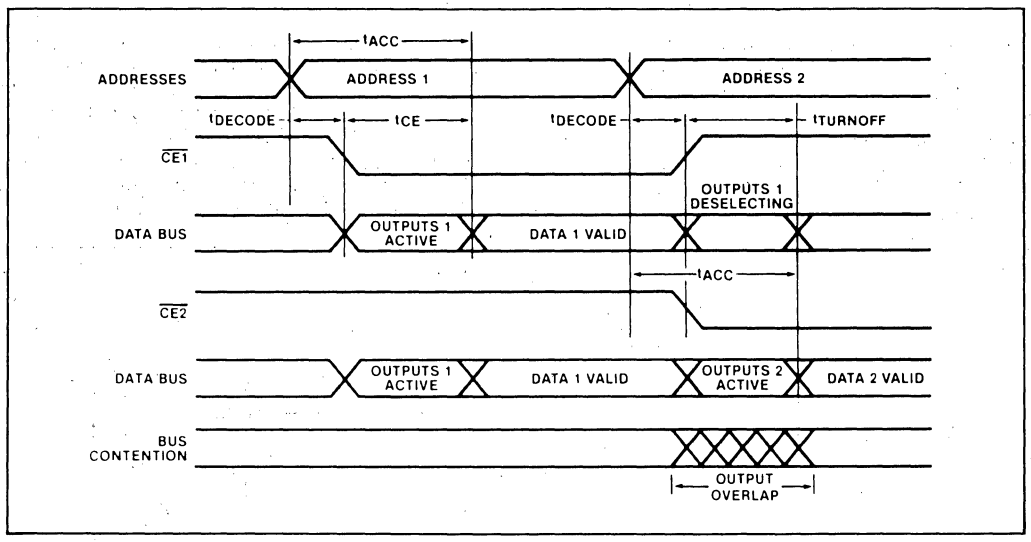


Figure 17. Single-Line Control and Bus Contention

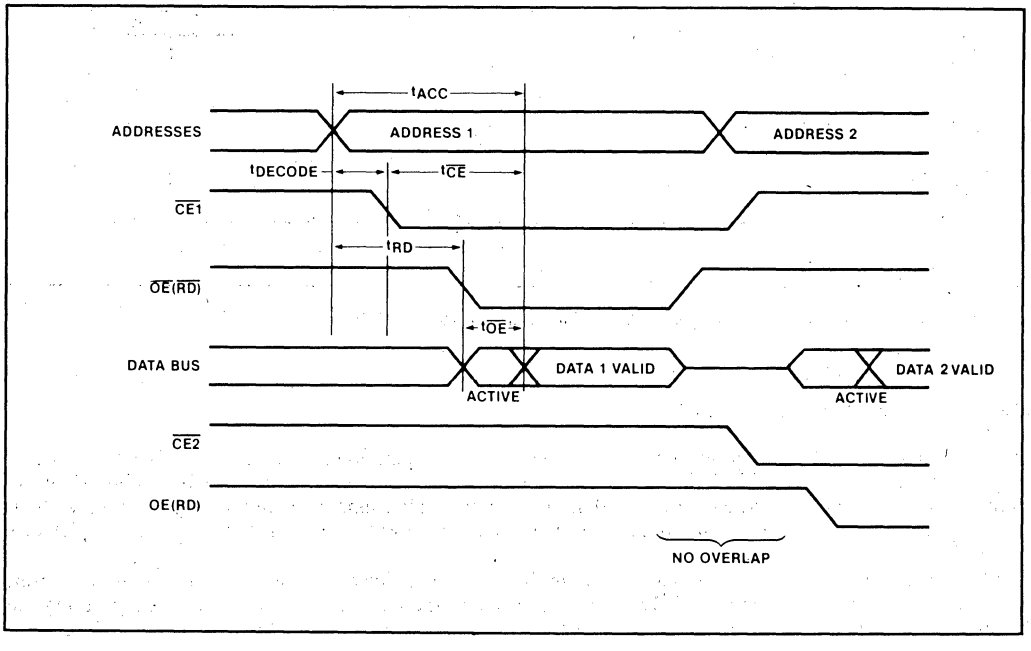


Figure 18. Two-Line Control Architecture

Interfacing to the 8088

Interfacing the 2817A to the 8088 8-bit processor is a simple task. The 8088 is a very powerful processor having a one megabyte direct addressing capability and the 8086 instruction set. Figure 19 shows the interface used on the 2817A/8088 applications demo board. The 8288 bus controller is used to generate various memory, I/O, and control signals. One of the signals is a delayed write pulse command, $\overline{MWTC}$, which is used to write to the 2186 Integrated RAM. The 8288 also acts as a buffer for all the memory, I/O, and control signals. As shown in Figure 19, two-line control is used in connecting the 8088 commands and address selection lines to the 2817A. The status of the $\overline{RDY}/\overline{BUSY}$ line is polled via the 8088's $\overline{TEST}$ input by using the WAIT instruction. See "How to use the $\overline{RDY}/\overline{BUSY}$ output" (below) for detailed operational information about the $\overline{RDY}/\overline{BUSY}$ line.

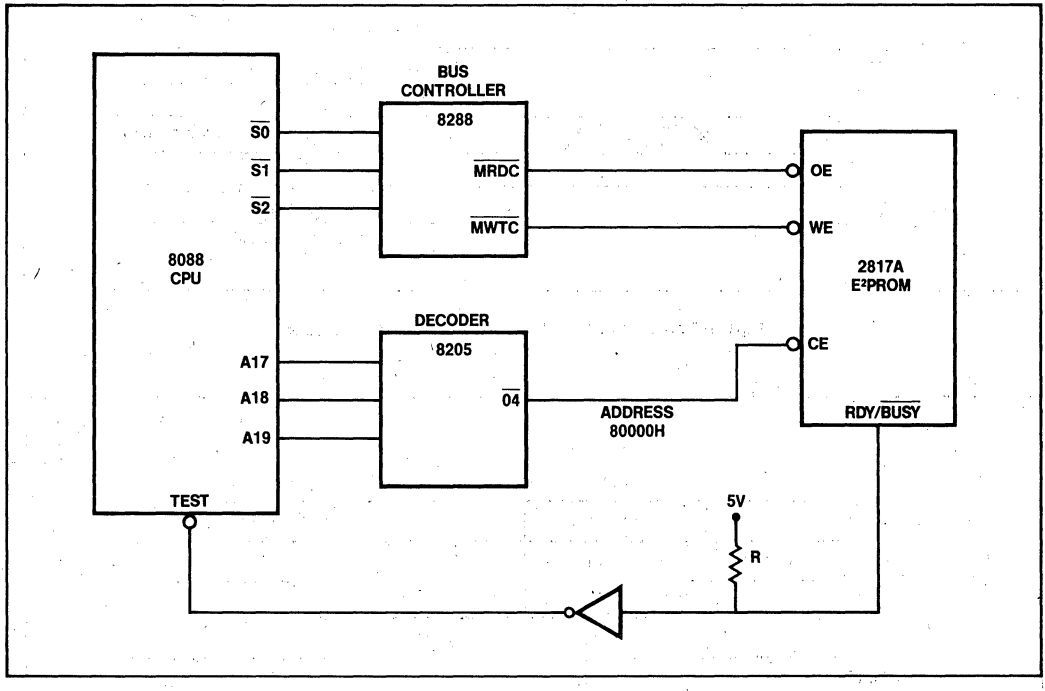


Figure 19. 2817A Interface to the 8088

DESIGN CONSIDERATIONS

How to use the $\overline{RDY}/\overline{BUSY}$ Output

The 2817A has a $\overline{RDY}/\overline{BUSY}$ output that indicates when a write operation is in progress and when the 2817A is ready for access. the $\overline{RDY}/\overline{BUSY}$ line goes low when a write operation starts and goes back high when the operation is completed. The system CPU can poll this output to determine when another byte can be written, or can use the output as an interrupt to notify the CPU when the current write operation has been completed.

Polled mode is usually the easiest interface to design. It is a good choice as long as the CPU can afford to wait while a byte is being written into the 2817A. In systems where the CPU must continue processing during the 20 milliseconds it takes to complete a byte write operation, an interrupt mode should be used.

Polled Methods

A polled mode can be used whenever it is not necessary for the CPU to be constantly processing while an E²PROM byte write operation is taking place. An example is the remote upgrade of a system's software over the telephone. The service center calls a customer and informs him that a new software package is available for the customer's system and is ready for transmission over the telephone. The customer connects the system to the phone line, either via an acoustic coupler or a direct connection, and waits about 40 seconds (see section on Fast Array Programming) while the new software package is being sent. During this time the only task the CPU is likely to have is the supervision of the downloading operation, so it can afford to wait while data is being written into E²PROM. The user's system now has the latest firmware package, received quickly and inexpensively over the phone, and good for another year or so until the next upgrade.

Polled mode is usually done by connecting the RDY/$\overline{\text{BUSY}}$ output to the CPU data bus via a three-state buffer as shown in Figure 20. (If desired, the rising edge of the RDY/$\overline{\text{BUSY}}$ signal could be used to set a positive edge-triggered flip-flop.) By polling the E2 RDY port the CPU can determine when the 2817A is ready for a read cycle or another write cycle. If any of the Intel I/O port devices (8155, 8255, 8355, or 8755A) are being used in the system, the RDY/$\overline{\text{BUSY}}$ output can be connected to one of the input ports of one of these devices.

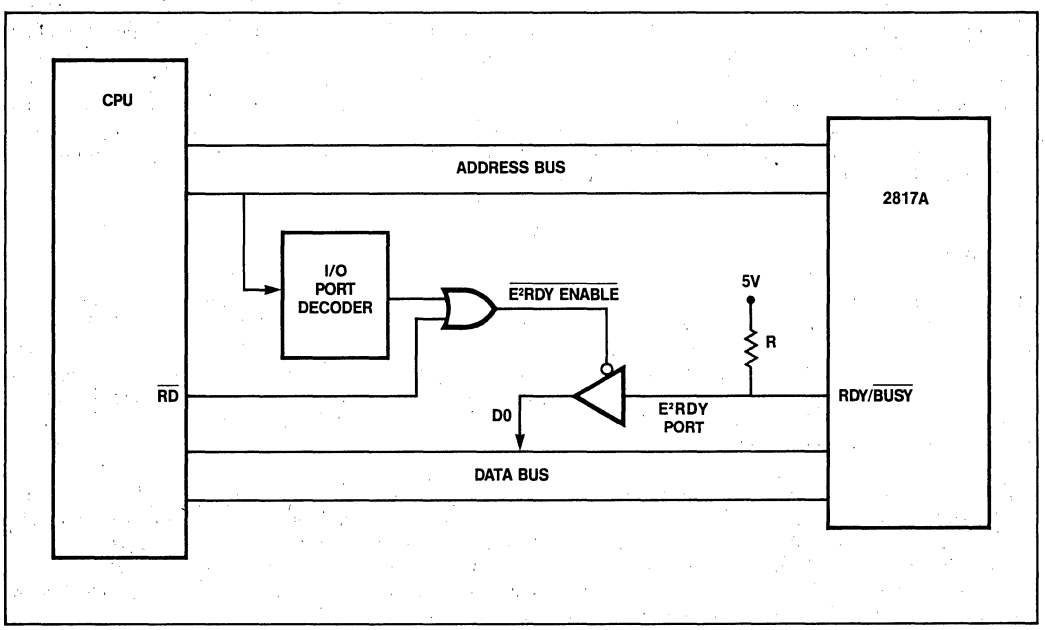


Figure 20. Polled Mode: Using an I/O Input Port to Poll the RDY/$\overline{\text{BUSY}}$ Output

Polled mode for the 8086/8088 microprocessor family is particularly easy using the $\overline{\text{TEST}}$ input and the WAIT instruction. This method is used on the 2817A/8088 demo board. The hardware hookup is shown in Figure 19.

In the 2817A/8088 demo board design a polled method was chosen for its hardware and software simplicity. The hardware consists of an inverter between the RDY/$\overline{\text{BUSY}}$ output and the $\overline{\text{TEST}}$ input of the 8088. In the software, writing to the 2817A simply requires a MOV instruction, and a NOP and a WAIT instruction, as follows:

```
;
;
        MOV     E2PROM, AL          ;THE AL REG HAS THE DATA BYTE TO
                                    :BE STORED
;
        NOP                         ;THIS ALLOWS ENOUGH TIME FOR THE
                                    :RDY/BUSY LINE TO BE RECOGNIZED
                                    :BY THE "WAIT" INSTRUCTION
;
        WAIT                        ;WAIT UNTIL THE WRITE OPERATION IS
                                    ;COMPLETED
;
;
```

The MOV instruction writes the data byte contained in the AL register to the 2817A. The 2817A latches the byte's address on the falling edge of the $\overline{\text{WE}}$ signal and latches the data on the rising edge of $\overline{\text{WE}}$. Inside the 2817A the write operation begins, while externally the RDY/$\overline{\text{BUSY}}$ output goes low. The inverted value to the 8088 $\overline{\text{TEST}}$ input is a "high". When the 8088 starts to execute the $\overline{\text{WAIT}}$ instruction the CPU will poll the $\overline{\text{TEST}}$ input. The short delay caused by the NOP instruction insures that the $\overline{\text{TEST}}$ input will be high when the wait instruction is executed. Otherwise, the $\overline{\text{TEST}}$ input would be sampled immediately after the write command ($\overline{\text{MWTC}}$) goes inactive high. As long as the $\overline{\text{TEST}}$ input is high the 8088 will remain in wait state. When the write operation in the 2817A has been completed, the RDY/$\overline{\text{BUSY}}$ output will go low, the $\overline{\text{TEST}}$ input to the 8088 will go high, and program execution will continue with the next instruction after WAIT.

Interrupt Methods

Interrupt mode is desirable when data or parameters need to be stored at irregular and relatively frequent intervals, and the CPU must be constantly processing. ("Relatively" meaning not so frequent as to exceed the write endurance of the E²PROM). With interrupt mode, as implemented in this application note, not only is the CPU's processing time not significantly affected by the E²PROM write cycles, but the user's main program does not have to worry about checking on the E²PROM to see if a given write cycle is done, nor do any status flags have to be monitored. The user program need only write the data to be stored in E²PROM to a RAM buffer table. The status and interrupt subroutines do the rest.

For the 8088, an 8259A Programmable Interrupt Controller is used to handle interrupt signals. The 8259A can itself handle up to 8 interrupt lines: each line is separately maskable and priority handling is dynamically programmable. The 2817A's RDY/$\overline{\text{BUSY}}$ output can be directly connected to any one of the 8259A's interrupt inputs. The edge-triggered mode is used to trigger the interrrupt input to the 8259A when the RDY/$\overline{\text{BUSY}}$ output returns high upon completion of the write operation. The schematic diagrams in Figure 21 and Appendix B show the electrical connections for using an 8259A with 8088 CPU.

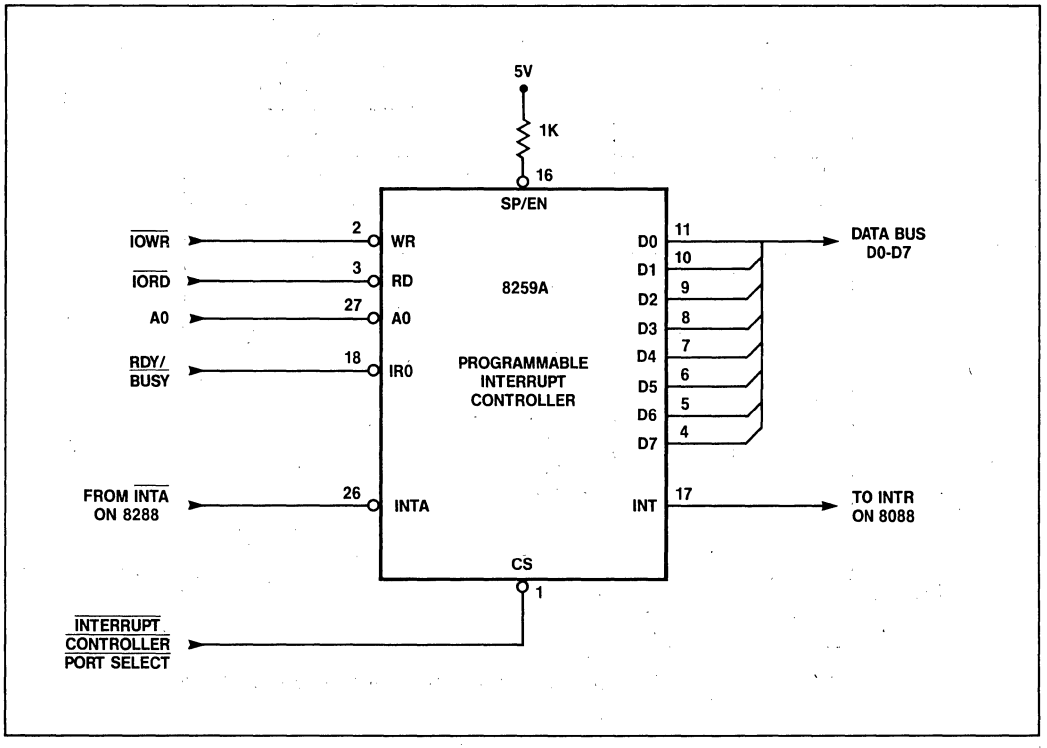


Figure 21. Interrupt Mode: 8259A in an 8088 System

Interrupt Subroutine

The following is an example of how to implement an interrupt-driven system in software.

A section of the system RAM is organized exactly like the look-up table in the 2817A E²PROM. In this example the table consists of ten bytes. (see Figure 22)

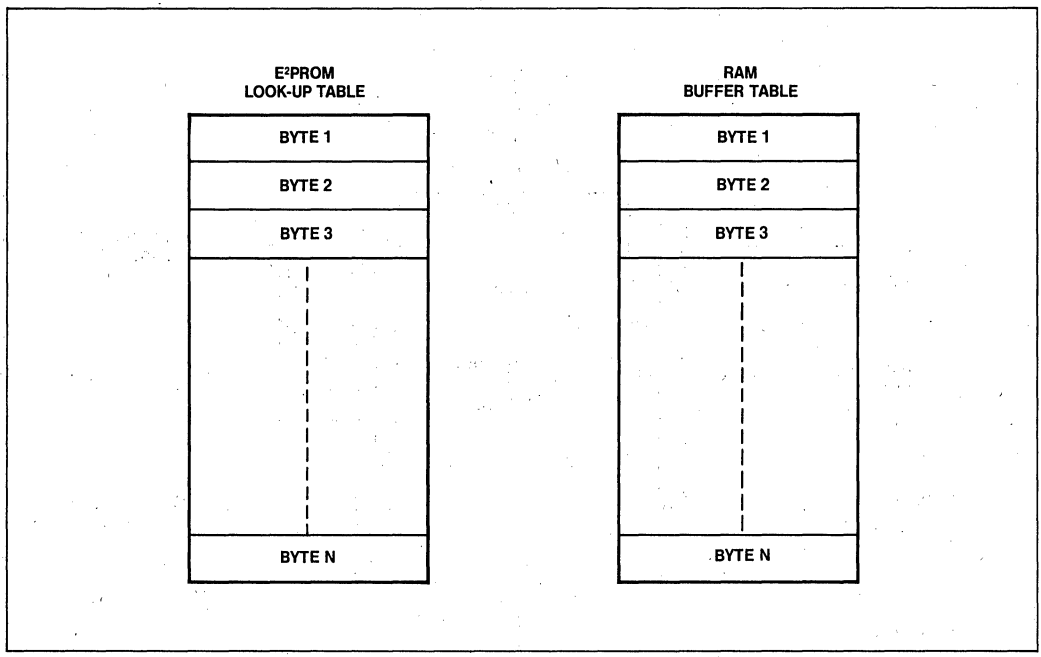


Figure 22. E²PROM Look-up Table and RAM Buffer for Interrupt Mode

Each time the user program receives or produces data to be stored in E²PROM, all the user program has to do is write the desired data into the corresponding locations in the RAM buffer table, then call a short status subroutine. The transfer of data to the E²PROM and the monitoring of the write operation are done by the status and interrupt subroutines.

The status subroutine checks if an E²PROM write operation is in progress, which would mean that the interrupt subroutine is already in the process of trasnsferring data from RAM to E²PROM. If so, then nothing else needs to be done. If no transfer is in progress, then the status routine will start an E²PROM write operation by looking for the first byte of data in the RAM buffer table that is not equal to its corresponding byte of data in the E²PROM table. When that write operation is complete, the interrupt subroutine is called, which checks for any more data to be written to E²PROM. The interrupt subroutine is called after each write operation is completed until all the new data in the RAM buffer table has been transferred to E²PROM.

The flowcharts for these subroutines are shown in Figures 23 and 24, and the software implementation for the 2817A/8088 demo board is on pages 44 through 47. The code to initialize the 8259A Programmable Interrupt Controller is also given in Appendix A.

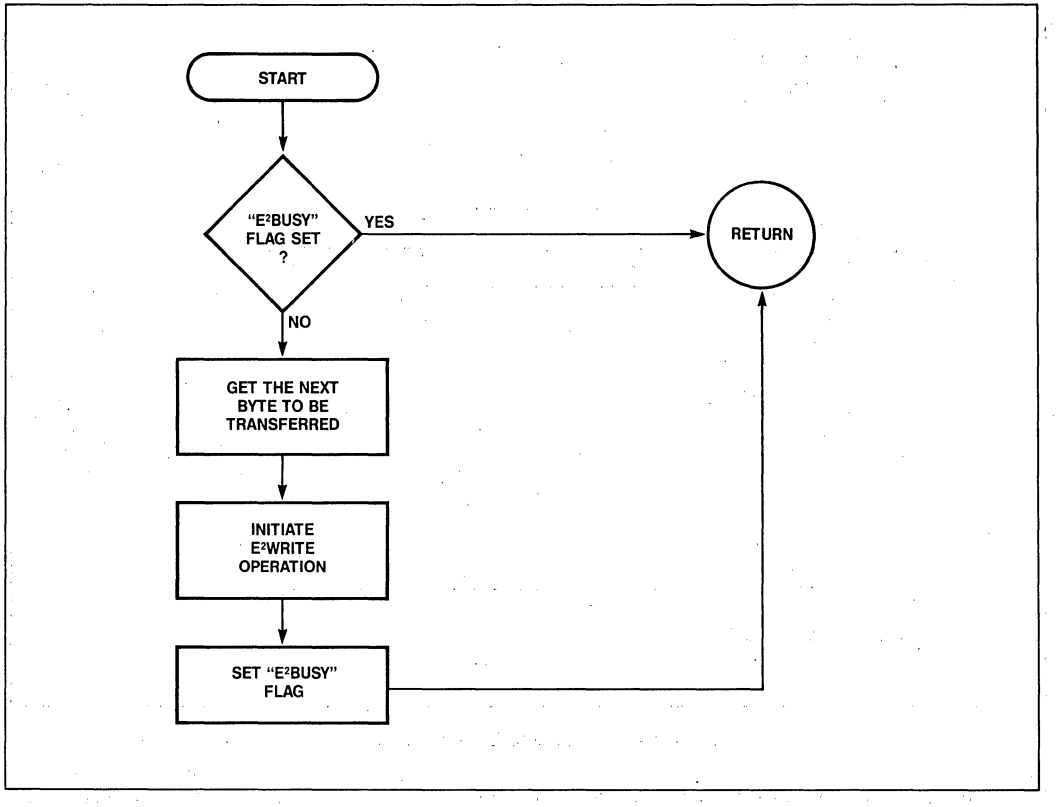


Figure 23. Status Subroutine

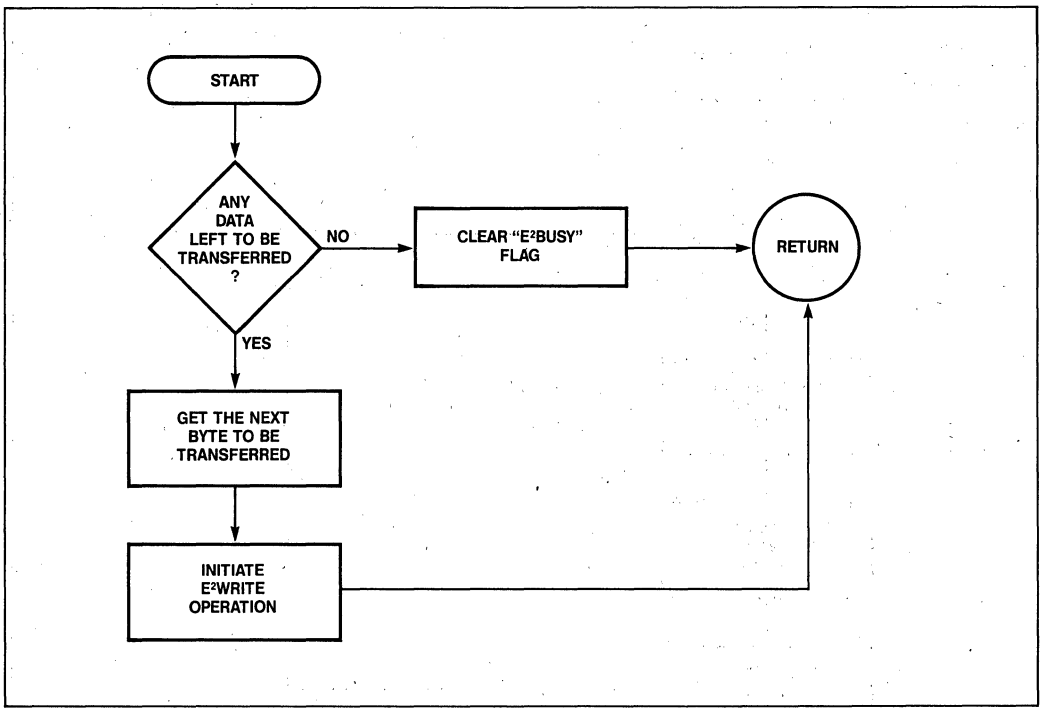


Figure 24. Interrupt Subroutine

When first initializing this look-up table, each byte in the E²PROM table is set equal to its corresponding byte in the RAM buffer table. Any subsequent changes to the RAM buffer table are easily detected by the status and interrupt subroutines by comparing each pair of corresponding locations in the two tables.

Combining Multiple RDY/$\overline{\text{BUSY}}$ Outputs in a 2817A Array

The 2817A's RDY/$\overline{\text{BUSY}}$ pin is an open-drain output, so multiple RDY/$\overline{\text{BUSY}}$ outputs can be or-tied together. The value of the pull-up resistor for the RDY/$\overline{\text{BUSY}}$ output can be calculated by the following formula:

$$R(\text{pullup}) = \frac{4.6V}{2.1ma - I_{IL}}$$

where I_{IL} (or I_{LI}) is the total V_{IL} input current of all device inputs connected to RDY/$\overline{\text{BUSY}}$.

There is no limit to the number of RDY/$\overline{\text{BUSY}}$ outputs that can be or-tied together.

Write Protection during Power Transitions

The 2817A has an on-chip write protection circuit that prevents a write operation from occurring when Vcc is below 4V.

All E²PROMs need inadvertent write protection, either internally or externally, when system power is being turned on or off. Since an E²PROM can be programmed in-circuit the possibility exists that it can be programmed accidentally when system power is removed. This is because the same control signals, $\overline{CE}$ and $\overline{WE}$, that can normally initiate a write operation could also trigger the write mode as system power is failing. As the 5V supply to the system falls, the system components will become unpredictable, and $\overline{CE}$ and $\overline{WE}$ may be spuriously driven active low as shown in Figure 25. To prevent a write operation from accidentally occurring when system power is dropping, an on-chip write protection circuit is needed to hold inactive at least one of the control signals. The same protection is needed when Vcc is rising from 0V to 5V as shown in Figure 25.

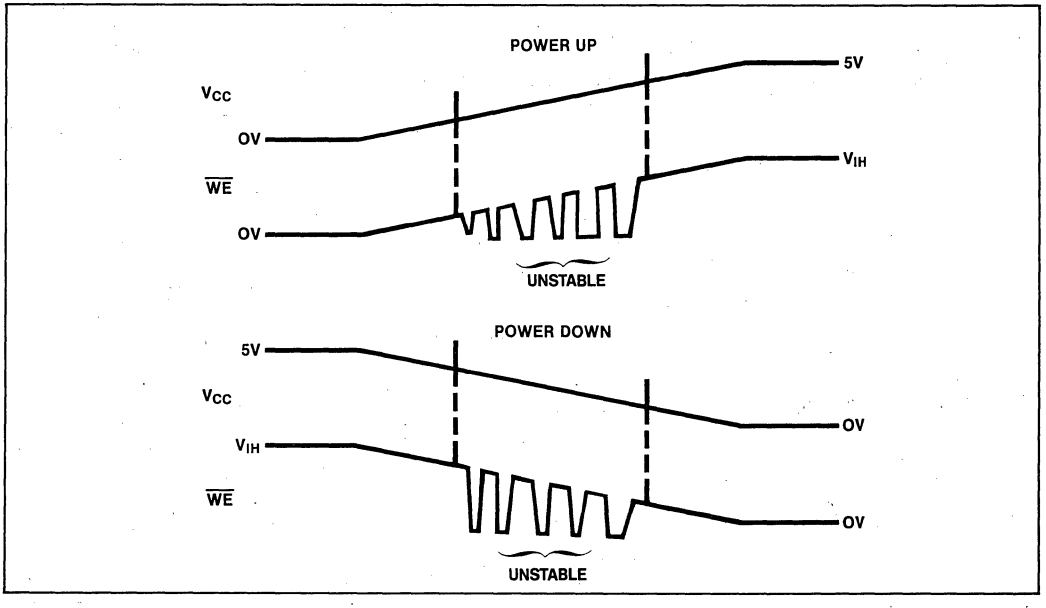


Figure 25. Typical TTL Driver Instability during V_{CC} Power Transitions

Alternative RDY/$\overline{BUSY}$ Interfacing Methods

There are many ways to interface the 2817A to a microprocessor using the RDY/$\overline{BUSY}$ output. The polled and interrupt-driven modes discussed in this application note are the easiest and most commonly used methods. These methods are compatible with the concept of an "intelligent" E²PROM which allows write operations to be done without holding up the microprocessor.

When first looking at the 2817A it might seem as if the RDY/$\overline{\text{BUSY}}$ line should be connected directly to a microprocessor's "ready" input since the signals have similiar names. There are certain timing and operational considerations, however, that make this method more difficult to design into a system than the interrupt or polled modes described earlier. These considerations are discussed below.

It is possible to use the RDY/$\overline{\text{BUSY}}$ output to hold up the microprocessor until the write operation is completed. This involves the insertion of wait states into the microprocessor's instruction cycle. For the 8085A and the 8088/8086, wait states occur when the READY input is brought inactive low. This type of interface would be implemented by connecting the 2817A's RDY/$\overline{\text{BUSY}}$ output to the microprocessor's READY input. At first this method might seem even simpler than polled mode, but a closer examination shows that this is not true.

The first consideration is the discrepancy between the timing of the 2817A's RDY/$\overline{\text{BUSY}}$ signal and the "ready" input of a microprocessor. Taking the 8088 as an example, the READY input is required to be low (if wait states are desired) about the same time that the 8088's write signal goes low. The top three signals in Figure 26 show this requirement. (The READY input shown is that of the 8284A clock generator. The output of the 8284A is then normally connected to the READY pin on the 8088. The $\overline{\text{AMWC}}$ write signal is the output of the 8288 bus controller which is used with the 8088 in MAX mode.) As shown by the bottom signal in Figure 26, the 2817A's RDY/$\overline{\text{BUSY}}$ output does not go low until after the $\overline{\text{WE}}$ input signal goes back high.

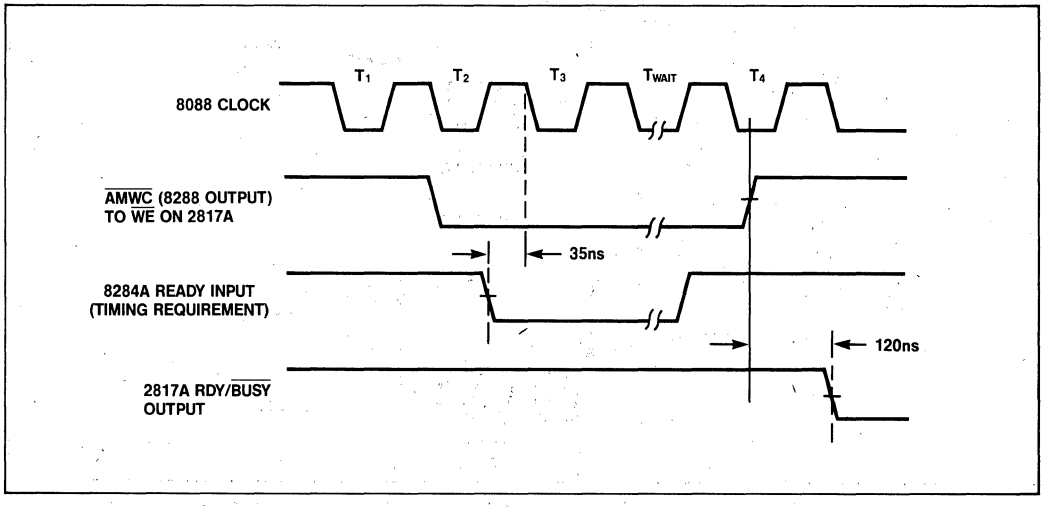


Figure 26. 8088 Ready Input Timing Requirement

Although this timing discrepancy is not critical, the implications for any given system (and the microprocessor used) must be examined carefully before this method of interfacing is used.

Another consideration that must be taken into account is the possibility of two or more sequential write cycles being done by the microprocessor. An 8088 string operation, for example, would allow the CPU to do one bus write cycle after another without any other intervening cycles. The 2817A would be able to respond successfully to the first write cycle, but not the second. The following would happen:

Upon seeing the rising and falling edges of the write pulse during the first write cycle, the 2817A would start an E²PROM write operation. The RDY/$\overline{\text{BUSY}}$ output would go low 75 ns after the rising edge of the write pulse as shown in Figure 27. The RDY/$\overline{\text{BUSY}}$ signal would not hold up the CPU until the following write cycle. While the first byte write operation is in progress the 2817A is not affected by any of the input signals and the data bus is in a high impedance state. When this write operation is completed and the RDY/$\overline{\text{BUSY}}$ signal returns high, the CPU will complete its second write cycle. The 2817A will not have recognized the second write cycle, however, because it did not see the $\overline{\text{WE}}$ signal fall.

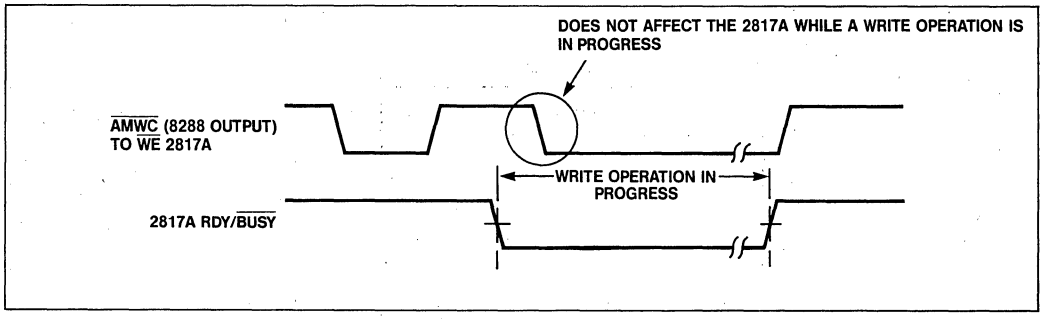


Figure 27. Limitation if 2817A RDY/$\overline{\text{BUSY}}$ is Connected to 8088 Ready Input: Loss of 2nd Byte During Two Consecutive Write Cycles

The direct connection of the 2817A's RDY/$\overline{\text{BUSY}}$ to the 8088's (or any other microprocessor's) READY input would thus require that no software be written that might cause two or more consecutive write cycles to occur.

The design of the 2817A's on-chip intelligence gives the 2817A maximum flexibility in interfacing to microprocessor-based systems. While the interrupt-driven and polled interface modes are two relatively simple (and widely used) approaches, many other methods are possible. In each case the timing specifications should be studied to insure that the timing requirements of all devices involved are met.

E²PROM Mapping Techniques

Inherent in the design of all E²PROM's is a 10,000 write cycle per byte limitation. As an example of the allowable write cycle frequency, all 2048 bytes in one 2817A could be written three times each day, 365 days a year, for 10 years before exceeding guaranteed limits. The E²PROM is not designed to be written to as often as RAM memory. Mapping techniques such as the ones shown here, however, allow higher write cycle frequencies than normal to be used.

Electronic Message Storage Applications

A memory usage mapping technique such as the one used in the Electronic Message Storage Demo described in this applications note can more evenly distribute the number of write cycles to each byte in the E²PROM array, thus stretching the usable life of the E²PROM. In message storage, if a pointer is used to indicate the location of the next empty message block, that pointer has to be re-written into an E²PROM memory location each time a message is stored in the E²PROM memory. That particular pointer storage location could reliably be used 10,000 times and still be guaranteed to remain within specification. If a system using this method had one 2817A for message storage, it could store 3 messages per day, 365 days a year for 10 years.

In the Electronic Message Storage Demo, no pointer is stored each time a message is stored. Instead, the 2817A is partitioned into a number of fixed-length message block spaces. Empty blocks have the byte "FF" in the 1st location, as shown in Figure 28. Thus, whenever a message is to be stored, the 2817A is searched for a block with an "FF" in the 1st location. Using this method, if the 2817A in a message storage system is divided into 20 100-byte blocks (2817A = 2K bytes), up to 60 messages can be stored each day, 365 days a year, for 10 years. In this example, for every additional 2817A in the system, 60 more messages of this size could be stored each day.

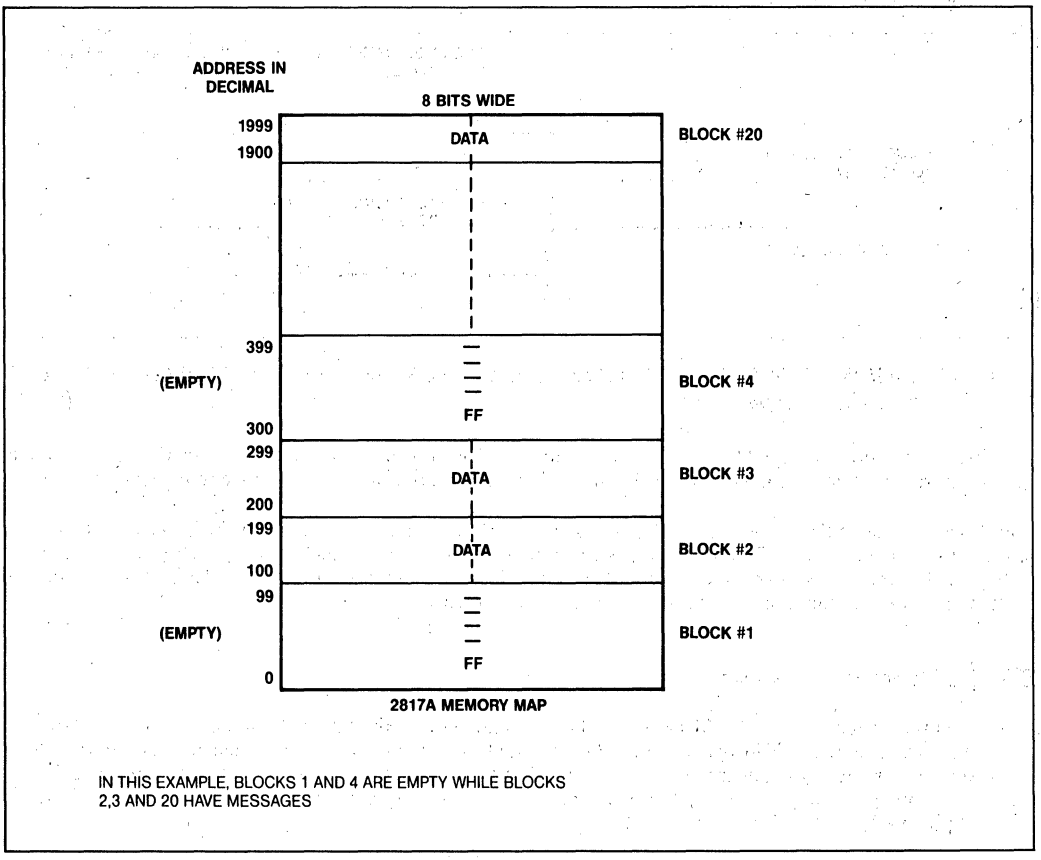


Figure 28. Message Storage Technique Example

Fast Array Programming

In large memory storage applications where fast write times are important, fast array programming can provide a solution. The total write time of any block of data to the array is reduced by a factor equal to the number of 2817As in the array. For example, an 8K byte block of data can be stored in an array of sixteen 2817As (32K x 8 of E²PROM memory) in 1/16 the normal time, that is, 8192 x 20 milliseconds x 1/16 = 10 seconds. This is particularly useful in designs that require a large amount of non-volatile memory to be used in harsh environments. No additional hardware or software is required: the memory address decoder is simply connected according to the schematic diagram in Figure 29. The result is a decoding of the least significant address lines rather than the most significant address lines.

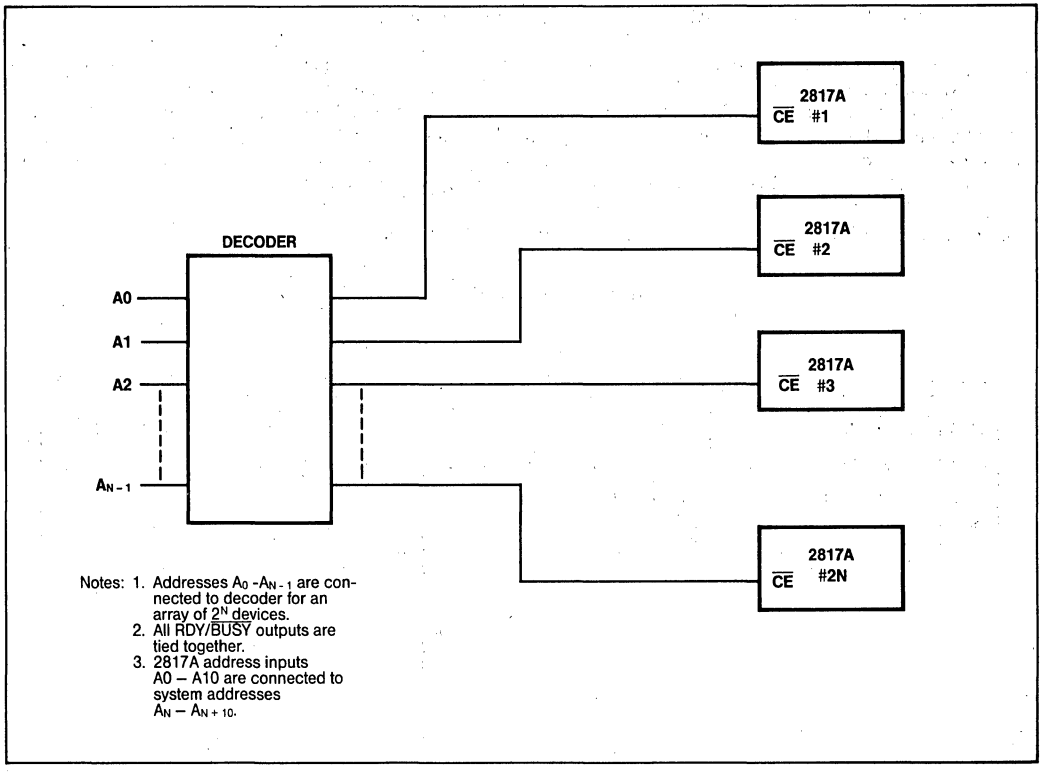


Figure 29. Fast Array Programming

To transfer data to the E²PROM memory, a series of high-speed, system write cycles are performed, one immediately after the other, to the E²PROM memory array. Each 2817A will internally perform a write operation using one of the successive bytes. For example, if the array in Figure 29 had 64 2817As, then 64 bytes would be written in each burst of high-speed write cycles. The total programming time for each 64-byte block would be the same as for one 2817A: 20 milliseconds.

Effectively, the total time needed to write a block of data into the array shown in Figure 29 is equal to:

Total Write Time =

$$\frac{\text{write time (20 ms) x \# of bytes to be stored}}{\text{\# of 2817A devices in the array}}$$

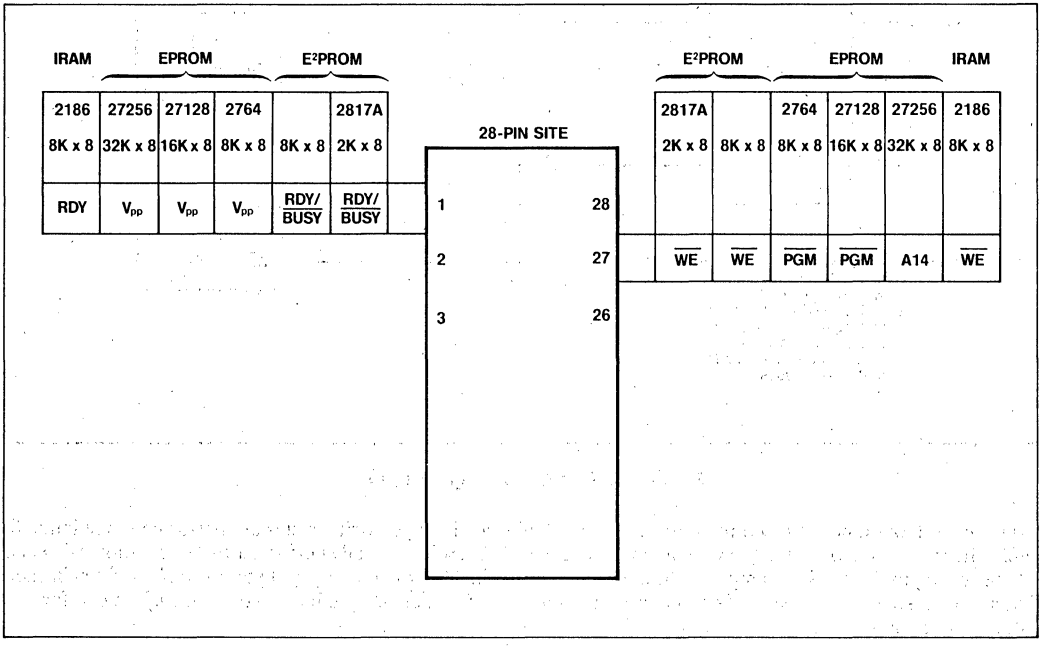
The more 2817As there are in the array, the shorter the write time for a given block of data. For example, writing a block of 16K bytes to an array of 32 2817As (64K bytes of E²PROM) would take 10 seconds (12K bits per second). A more significant application would be a ¼ megabyte E²PROM array (128 2817As). Writing 100K bytes to such an array would take only 15 seconds, which is 51K bits/second.

MULTI-DEVICE CONFIGURATION ON THE 28-PIN SITE

Intel's line of 28-pin memory devices is designed to be easily interchangeable on a single 28-pin site. The hardware and software designer need no longer bear the risk of board re-design due to inaccurate estimations of the amount and types of memory required. E²PROMs, EPROMs, and RAMs can be mixed and matched in the same 28-pin site. A small number of jumpers can be used to accommodate the different signals that use pins 1 and 27. This section has examples of how to lay out a printed circuit board memory array to accommodate any combination of 28-pin memories, including the multi-device sites on the 2817A/8088 applications demo.

The chart in Table 1 shows the different signals for pins 1 and 27 for Intel's 28-pin memories.

Table 1. 28-Pin Memory Device Compatibility



2817A/8088 Applications Demo Multi-Device Sites

Figure 30 shows a pin matrix of nine pins that can be used to allow different types of 28-pin memory devices to be plugged into socket U 18 on the 2817A/8088 applications demo board. A 27128 UV EPROM is normally plugged into socket U18 in Figure 30. A 2817A E²PROM is plugged into socket U17, and a 2186 iRAM resides in socket U19. Figure 31 shows which jumper pairs to connect for any one of the following devices in socket U18:

2817A	2K x 8 E²PROM
	8K x 8 E²PROM
2764	8K x 8 UV EPROM
27128	16K x 8 UV EPROM
27256	32K x 8 UV EPROM
2186	8K x 8 iRAM

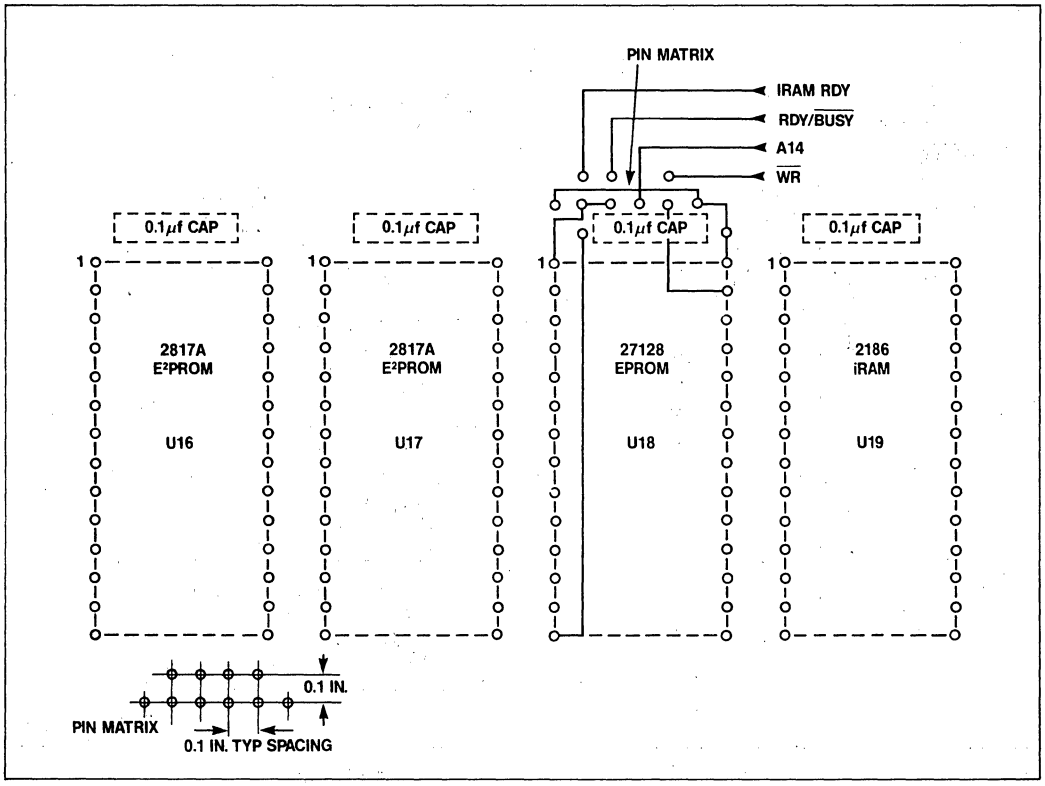


Figure 30. 2817A/8088 Applications Demo 28-Pin Site Array

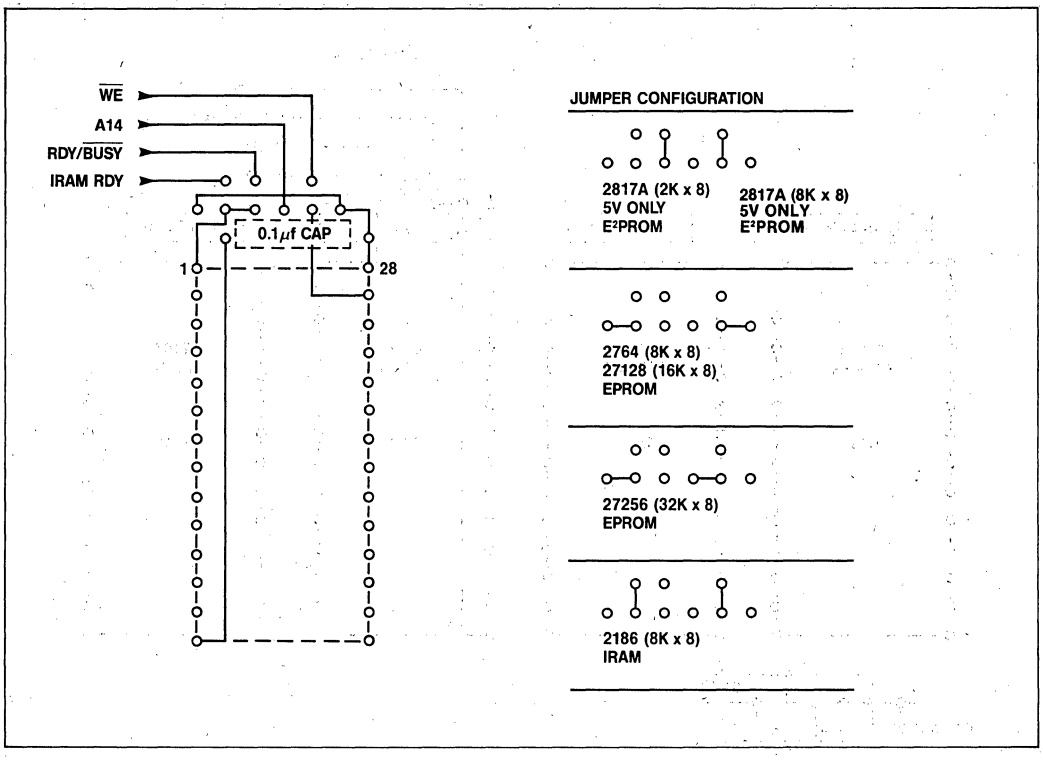


Figure 31. 2817A and 28-Pin EPROM/iRAM Site

If three 2817A's are plugged into the board in sockets U16, U17, and U18, a new firmware package can be remotely downloaded via the comline interface. The program code for performing the downloading operation would reside in the 2817A in socket U18. The routine in this E²PROM would perform most of the downloading operation. After the 2817As in sockets U16 and U17 have been loaded, program control would be transferred to one of the E²PROMs in those sockets in order to load the 2817A in socket U18 (see also "Application: Firmware Remote Downloading" on page 3).

By designing with the JEDEC-compatible 28-pin site and using a small number of jumbers as shown in the examples, boards and systems can be designed to be highly flexible and to have a long product life.

2817A/8088 APPLICATIONS DEMO BOARD CIRCUIT OPERATION

The memory map in Figure 32 shows the organization of the memory space in this design. The input/output map in Figure 33 shows the addresses of the I/O ports. Figure 34 shows a simple block diagram of the 2817A/8088 demo board. Figure 35 contains a more detailed block diagram. The complete schematic for the board is in Appendix B.

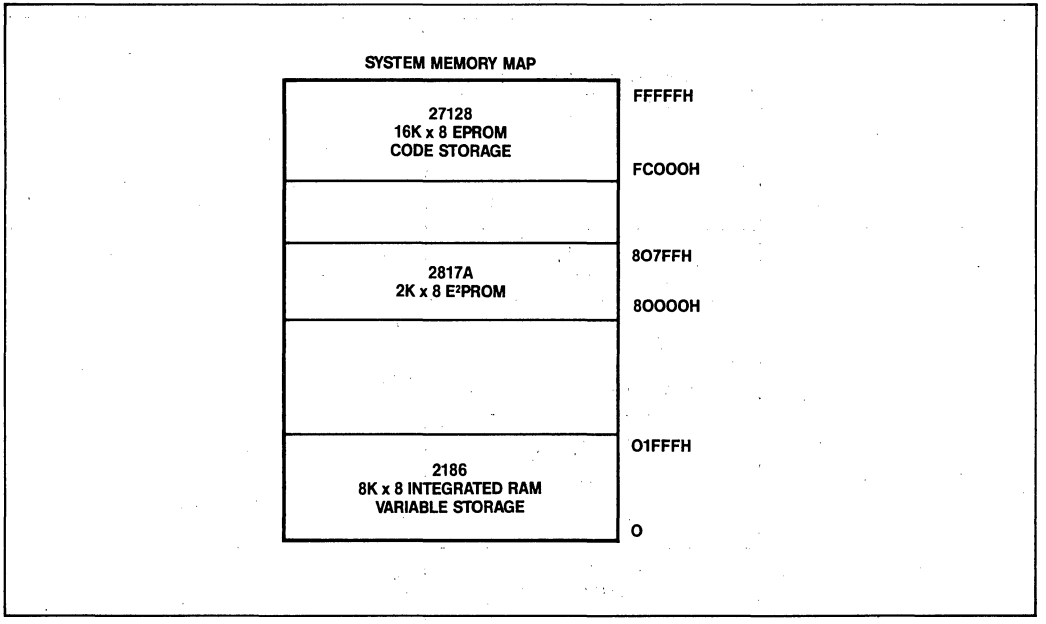


Figure 32. 2817A/8088 Applications Demo Board

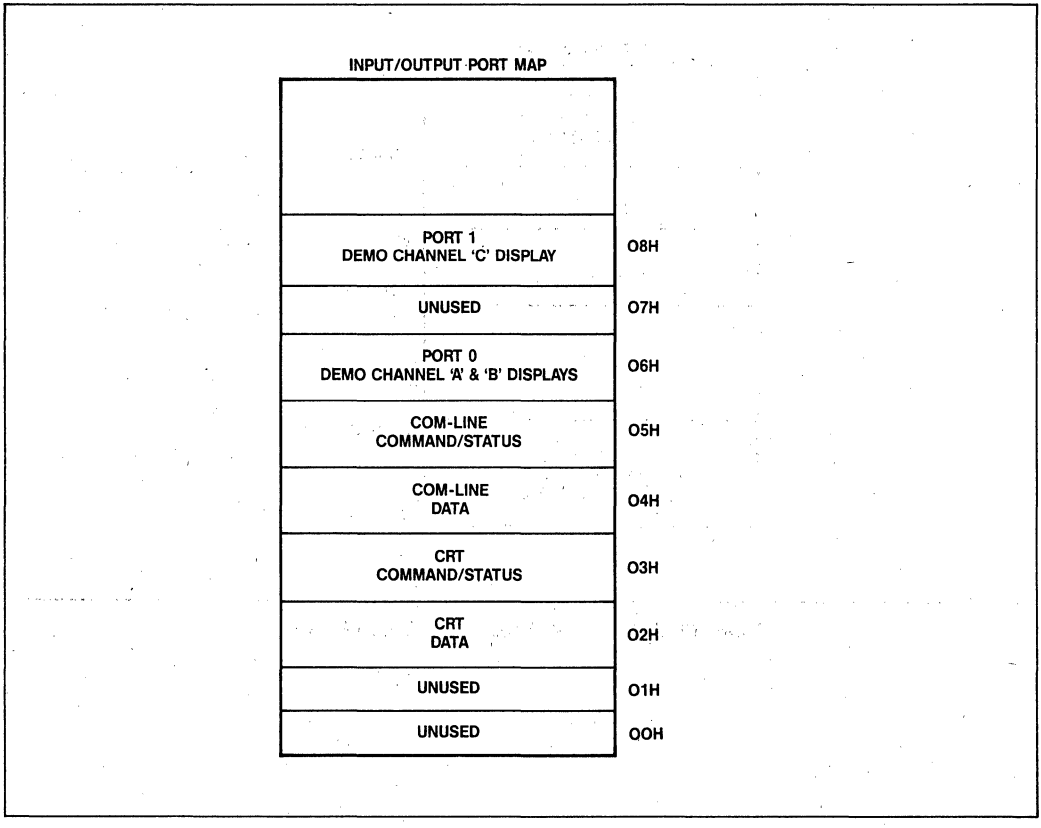


Figure 33. 2817A/8088 Applications Demo Board

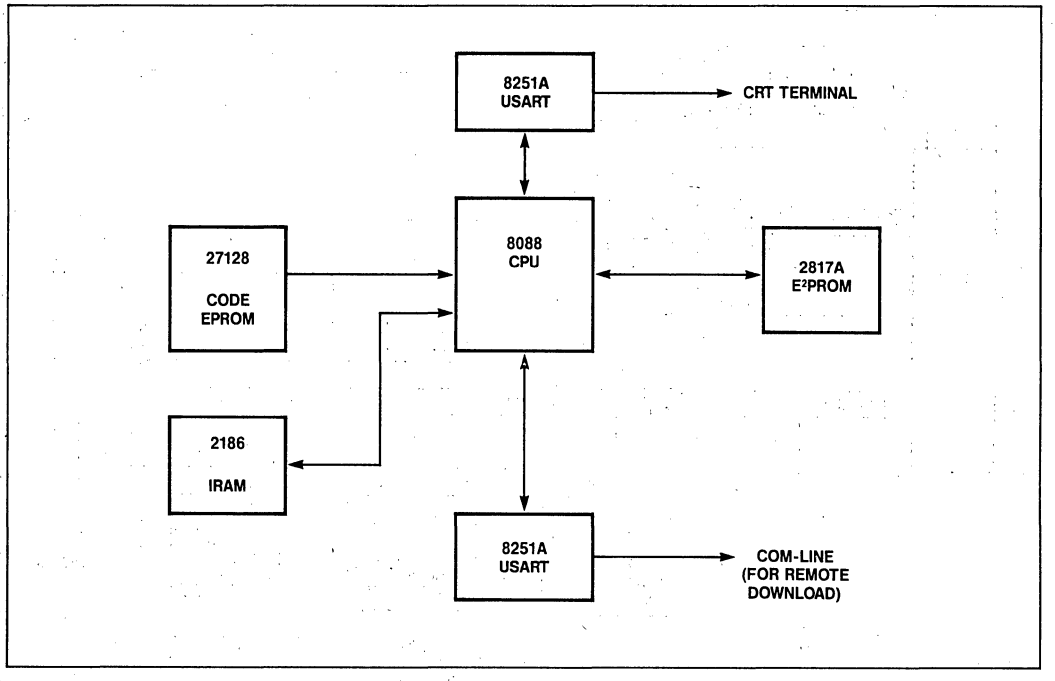


Figure 34. 2817A/8088 Applications Demo Board Simplified Block Diagram

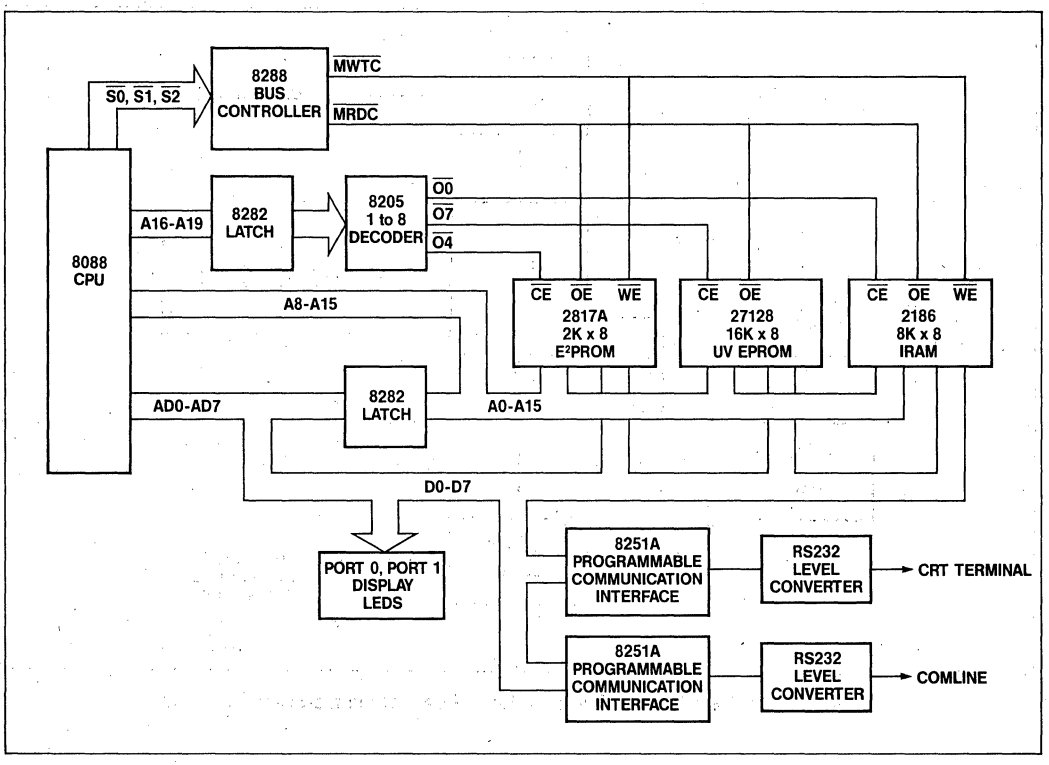


Figure 35. 2817A/8088 Applications Demo Board: Full Block Diagram

2817A Interface Circuitry: The RDY/$\overline{\text{BUSY}}$ Output (Figures 37 and 38)

The 2817A's RDY/$\overline{\text{BUSY}}$ output is connected to the $\overline{\text{TEST}}$ input of the 8088 after being inverted. This allows the 8088 CPU to go into a wait mode via the WAIT instruction while 2817A write operations are in progress. For more detailed information, see "How to Use the RDY/$\overline{\text{BUSY}}$ Output" on page 19.

CPU: The 8088 8-bit Microprocessor (Figure 37)

The 8088 microprocessor has an 8-bit wide external data bus and a one megabyte direct addressing capability. It also has an address/data multiplexed bus, on which the 8-bit data bus is multiplexed with the lower 8 bits of the address bus. The 8088's internal data/address paths are all sixteen bits wide. It has a full range of arithmetic capabilities including multiplication and division, all operating directly on sixteen-bit quantities. All of the arithmetic operations can be performed on any one of the eight 16-bit general purpose registers. The 8088 also has string operation commands which allow high-speed processing of large data blocks.

MEMORY

2817A-2K x 8 E²PROM (Figure 38)

The 2817A is used to store user messages entered from the keyboard and sent over the comline channel from other demo boards. The 2817A also stores the maintenance log information, operator's initials entered from the keyboard, and the system re-configuration demo status bytes.

27128-16K x 8 UV EPROM (Figure 40)

The 16K byte demo program, which consists of 5K bytes of 8088 code and 11K bytes of ASCII text data, is stored in the 27128. This 16K byte EPROM is located at the top of the 8088's memory space as shown in Figure 32.

2186-8K x 8 Integrated RAM (Figure 40)

This device has a dynamic RAM memory array with on-chip refresh controller circuitry. The stack, buffer space, and variables for the 8088's program code are stored in the 2186. The only external circuitry required is the $\overline{CE}$ pulse forming circuit shown in Figure 40. This circuit generates a single, glitchless $\overline{CE}$ pulse each time the iRAM memory is to be accessed. The operation of the $\overline{CE}$ pulse circuit is as follows: the iRAM $\overline{CE}$ output will not go active low until both the memory space of the iRAM is selected and the ALE pulse occurs. The iRAM $\overline{CE}$ signal will then stay low until the iRAM's READY output goes back high and the 74S112 flip-flops are cleared on the falling edge of the CLK signal.

Demo Channel Display LED Ports (Figure 42)

Two 74LS374 8-bit D flip-flop devices are used to store the system re-configuration demo status bits for display with LEDs. Output ports 0 and 1 are used to control nine light-emitting diodes. The output states of these bits are determined by the system re-configuration look-up table in E²PROM (see Figure 14). When the demo board is powered up, the firmware reads the look-up table to determine what states the port bits should be in. These port bits could easily control peripheral channelling between systems and peripheral devices in a computer room. The port bits could also control system parameters such as communications modes, baud rates, display formats, and "soft key" definitions.

8251A Programmable Communications Interface

Hardware (Figure 41)

Baud Rate

The first counter in the 74LS393 is used to generate the 4800 baud rate for the CRT terminal comm(communications)-interface. The baud rate is produced by dividing the PCLK 2.5 MHZ clock by 8 to obtain a 312.5KHZ clock. The 2nd 4-bit counter in the 74LS393 is used to genrate the 300 baud rate for the Comline comm-interface. This is done by dividing the 312.5KHZ clock by 16 down to 19.5KHZ.

RS232 Level Interface Circuit

A standard RS232 serial line driver and receiver circuit requires ±12V power supplies to generate the required ±12V signal levels. To reduce the complexity of operating this demo the RS232 level shifting circuits were designed so that they need only a +5V power supply. These circuits allow the demo board to communicate with any system that has standard RS232 level inputs and outputs. The level shifter circuit shown with the CRT interface, however, will not actively assert a valid logic "0" level when connected to another level shifter exactly like it. To allow two demo boards to communicate with each other the level shifter circuit at the top of Figure 41 has to be modified. For this reason the Comline level shifter at the bottom of Figure 41 is slightly different than the CRT interface level shifter at the top of the Figure. This allows the Comline channels of two boards to be connected together so that data can be transferred between demo boards to be ultimately stored in E²PROM.

An example of a circuit that will provide the full RS232 spec + and -12V signal levels is shown in Figure 36.

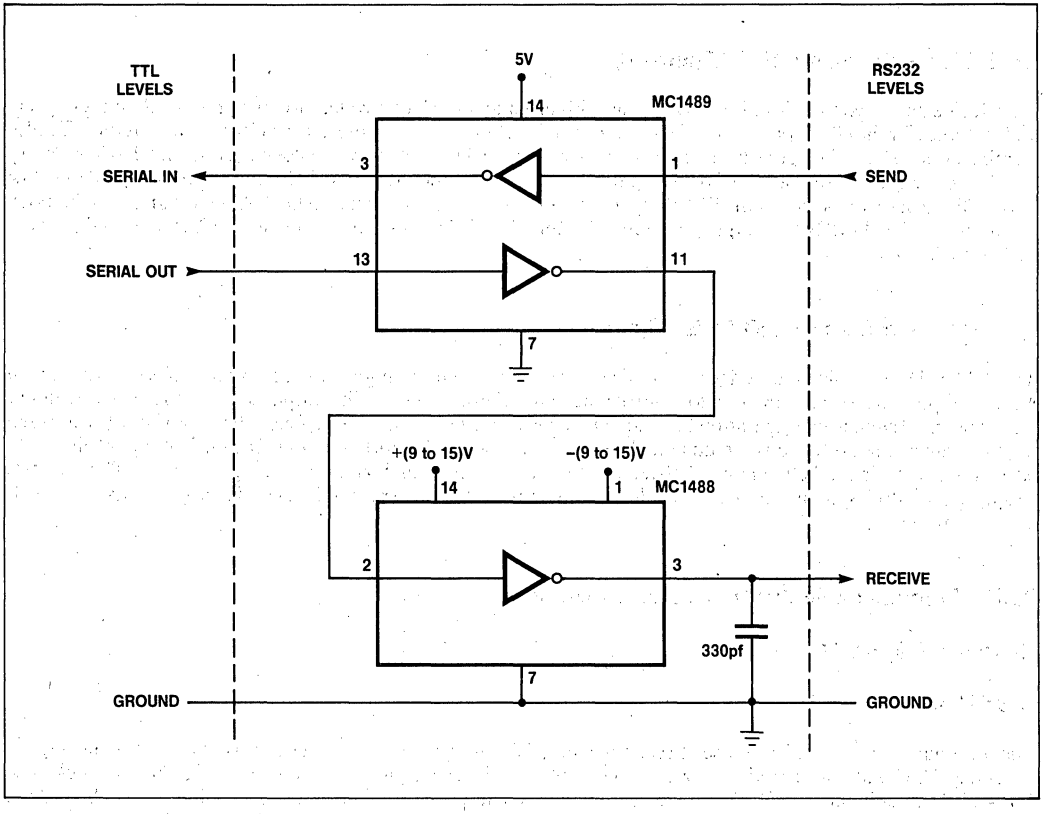


Figure 36. Full Spec TTL to RS232 Level Converter Circuit

8251A Software Initialization

The following software listing shows the procedure used on the 2817A/8088 applications demo to initialize the 8251A Programmable Communication Interface devices.

```
;
;
;
;
;            EQUATES
;
;
MODE_INSTR        EQU         11001111B         ;ASYNCHRONOUS MODE
                                                ;64X BAUD RATE
                                                ;8-BIT CHARACTER LENGTH
                                                ;PARITY DISABLED
                                                ;2 STOP BITS
;
COMMAND_INSTR     EQU         00100101B         ;TRANSMIT ON
                                                ;DTR/ = 1
                                                ;RECEIVE ENABLE


                                                ;NORMAL OP - NO BREAK CHAR
                                                ;NO ERROR RESET
                                                ;RTS/ = 0


;
CRT_PCI_CMD             EQU     03H     ;COMMAND MODE I/O PORT FOR
                                        ;THE CRT TERMINAL PROG'BLE COMM
                                        ;INTERFACE
;
COMLINE_PCI_CMD         EQU     05H     ;COMMAND MODE I/O PORT FOR
                                        ;THE COMLINE USART
;
;


;
;       CODE TO INITIALIZE THE 8251A'S
;
;
        MOV         DX,CRT_PCI_CMD
        CALL        INIT_PCI            ;INITIALIZE THE CRT COMM-INTERFACE
;
        MOV         DX,COMLINE_PCI_CMD
        CALL        INIT_PCI            ;INITIALIZE THE COMLINE PCI
;
;
;
;
```

```
;
;*************************************************************************
;
;               SUBROUTINES
;
;
;
INIT_PCI:
;
                                                ;INITIALIZE THE COMM-INTERFACE
                                                ;DEVICE SPECIFIED
                                                ;IN THE 'DX' REGISTER.

;
;
        MOV     AL,MODE_INSTR
        OUT     DX,AL
;
        MOV     AL,COMMAND_INSTR
        OUT     DX,AL
;
        RET
;
;
```

Support Devices

8284A-The Clock Generator (page 50)

This device produces a clean, stable clock for the 8088 with the correct 33% duty cycle.The 8284A also interfaces the system's asynchronous iRAM READY signal with the 8088, as well as generating the RESET signal.

8288-Bus Controller (page 52)

This device generates the memory access signals (memory read, memory write, ALE, etc.) for the system.

APPENDIX A
SOFTWARE LISTINGS

Routines for Data Transfer Between 2817A/8088 Demo Boards over the 'COMLINE' Communications Interface Channel

This is the routine for the system sending the data block.

```
;
;
COMLINE_PCI_DATA        EQU     4H      ;8251A PROGRAMMABLE COMMUNICATIONS
                                        ;INTERFACE DATA PORT
;
COMLINE_PCI_CMD         EQU     5H      ;PROG'BLE COMM-INTERFACE COMMAND/
                                        ;STATUS PORT
;
;
;
;
        MOV     AH,SOT
        CALL    SEND_CHAR               ;SEND A 'START OF TRANSMISSION' CHARACTER
                                        ;(THE OTHER SYSTEM WILL RECEIVE THE 'SOT'
                                        ;CHARACTER AND RETURN AN ACKNOWLEDGEMENT
                                        ; CHARACTER)
;
        CALL    REC_CHAR                ;GET A CHARACTER FROM THE OTHER SYSTEM
        CMP     AH,ACK                  ;IS IT AN 'ACKNOWLEDGE' CHARACTER?
        JNE     NOT_READY               ;IF NOT, INFORM OPERATOR THAT THE
                                        ;OTHER SYSTEM IS NOT READY.
;
        CALL    DELAY                   ;SMALL DELAY TO LET THE OTHER GET READY
;
        LEA     BP,DATA_BLOCK           ;LOAD POINTER TO DATA BLOCK
SEND_LOOP:
        MOV     AH,[BP]                 ;FETCH A BYTE
        CALL    SEND_CHAR               ;SEND IT
        CMP     AH,EOS                  ;END OF DATA STRING?
        JNE     SEND_LOOP               ;IF NOT, CONTINUE
;OTHERWISE, FINISH UP
```

Next, the software for the system receiving the data block:

```
;
        CALL    REC_CHAR                ;GET CHARACTER FROM THE COMLINE
        CMP     AH,SOT                  ;START OF TRANSMISSION CHAR?
;
        JE      CONT_1                  ;IF SO, CONTINUE
        JMP     MAIN_LOOP               ;IF NOT, RETURN TO MAIN LOOP
;
CONT_1:
        MOV     AH,ACK
        CALL    SEND_CHAR               ;SEND AN ACKNOWLEDGEMENT CHARACTER
                                        ; BACK TO THE OTHER UNIT
```

```
;
        LEA     BP,DATA_BLK_BUFFER          ;SET POINTER TO DATA BLOCK
                                            ;BUFFER
;
REC_BLK_LOOP:
        CALL    REC_CHAR            ;GET BYTE FROM THE COMLINE
        CMP     AH,EOS              ;END OF DATA BLOCK?
        JE      FINISH_UP           ;IF SO, FINISH UP
        MOV     [BP],AH             ;IF NOT, STORE BYTE AND GET ANOTHER
        JMP     REC_BLK_LOOP
;
FINISH_UP:
;
```

The following are the subroutines used for the software for data transfer between 2817A/8088 Demo boards.

```
;
;               SUBROUTINES
;
;
;
;
SEND_CHAR:                          ;TAKES THE BYTE IN THE 'AH' REG. AND SENDS
                                    ; IT OVER THE COM-LINE TO THE OTHER SYSTEM.
;
;
RDY_SEND_POLL:
        IN      AL,COMLINE_PCI_CMD
        TEST    AL,1                        ;READY TO SEND YET?
        JE      RDY_SEND_POLL               ;IF NOT, CHECK AGAIN
;
        MOV     AL,AH
        OUT     COMLINE_PCI_DATA            ;IF READY, SEND CHARACTER
        RET
;
;
;
;
REC_CHAR:                           ;GETS A BYTE FROM THE OTHER SYSTEM OVER
                                    ;THE COM-LINE(VIA THE COM-LINE COMM-
                                    ;INTERFACE) AND
                                    ;PASSES IT BACK TO THE CALLING ROUTINE IN
                                    ;THE 'AH' REG.
```

```
;
;
RDY_REC_POLL:
        IN      AL,COMLINE_PCI_CMD
        TEST    AL,2                    ;HAS A CHARACTER BEEN RECEIVED YET?
        JE      RDY_REC_POLL            ;IF NOT, TRY AGAIN
;
        IN      AL,COMLINE_PCI_DATA     ;IF READY, GET CHARACTER
        MOV     AH,AL                   ;SAVE IT
        AND     AH,7FH                  ;STRIP OFF THE PARITY BIT
        RET
;
;
```

Interrupt Mode Software

Status Subroutine

```
;
;
XFER_STATUS_SUBR:                       ;DETERMINES IF AN E2PROM TRANSFER IS
                                        ;ALREADY IN PROGRESS.  IF NOT, THE
                                        ;NEXT BYTE TO BE TRANSFERRED IS
                                        ;WRITTEN TO E2PROM.
;
;
        CMP     E2_BUSY,1               ;E2 WRITE IN PROGRESS?
        JNE     DO_TRANSFER             ;IF NOT, PROCEED
        RET                             ;IF SO, RETURN, SINCE THE NEW DATA
                                        ;IN THE RAM BUFFER TABLE WILL BE
                                        ;DETECTED BY THE INTERRUPT SUBROUTINE
                                        ;AND EVENTUALLY TRANSFERRED TO
                                        ;E2PROM.
;
DO_TRANSFER:
        CALLWRITE_E2_DATA               ;FIND NEXT BYTE TO BE TRANSFERRED AND
                                        ;WRITE IT TO THE E2PROM.
;
;
        RET
;
;
```

Interrupt subroutine

```
;
;
E2_INTERRUPT_SUBROUTINE:                ;EACH TIME AN E2PROM WRITE OPERATION
                                        ;ENDS, CHECK IF THERE ARE ANY MORE
                                        ;BYTES TO BE TRANSFERRED.  IF SO,
                                        ;FETCH THE BYTE AND WRITE IT TO
                                        ;THE E2PROM.
;
        PUSH    DX
        PUSH    AX
        MOV     AL,OCW2
        MOV     DX,PIC_A0_LOW           ;INFORM THE PROGRAMMABLE
        OUT     DX,AL                   ;INTERRUPT CONTROLLER THAT THE
        POP     AX                      ;INTERRUPT HAS BEEN SERVICED
        POP     DX
;
        CALL    WRITE_E2_DATA
        IRET
;
```

This subroutine determines if there is any data left to be transferred from the RAM Buffer table to the E²PROM table, and if there is, transfers the next byte.

```
;
;
WRITE_E2_DATA:                          ;CHECK FOR ANY MORE DATA TO BE
                                        ;TRANSFERRED.  IF NONE, CLEAR THE
                                        ;"E2_BUSY" FLAG.  IF ANOTHER BYTE
                                        ;IS TRANSFERRED, SET THE "E2_BUSY"
                                        ;FLAG.
;
;
        PUSH    SI
        PUSH    DI
;
        LEA     SI,RAM_BUFFER_TABLE     ;POINT THE SI REG. TO THE RAM
                                        ;BUFFER TABLE
;
        LEA     DI,E2PROM_TABLE         ;POINT THE DI REG. TO THE E2PROM
                                        ;LOOK-UP TABLE
;
        MOV     CX,10                   ;DEFAULT COUNT = SIZE OF LOOK-UP
                                        ;TABLE
;
REPE    CMPSB                           ;SEARCH FOR ANY NEW DATA BYTES
                                        ;IN THE RAM BUFFER TABLE.
;
        JZ      NO_NEW_DATA             ;IF NO NEW DATA, FINISH UP
;
        ;OTHERWISE, WRITE THE NEW DATA BYTE TO E2PROM
        DEC     SI
        DEC     DI                      ;BACK UP TO THE NEW, UNEQUAL BYTE
;
        MOVSB                           ;WRITE THE NEW BYTE TO THE 2817A
                                        ; (THE 2817A WILL NOW START A WRITE
                                        ;  OPERATION, INTERNALLY)
```

```
;
        MOV     E2_BUSY,1               ;SET THE FLAG
;
        JMP     E2_XFER_FINISH



NO_NEW_DATA:
        MOV     E2_BUSY,0               ;CLEAR THE FLAG
E2_XFER_FINISH:
        POP     DI
        POP     SI
        RET
;
;
```

This is how a section of the stack segment is allocated for the RAM Buffer table.

```
;
;
STACK     SEGMENT
;
;
;
RAM_BUFFER_TABLE    DB      ?       ;BYTE 1
                    DB      ?       ;BYTE 2
                    DB      ?       ;BYTE 3
                    DB      ?       ;BYTE 4
                    DB      ?       ;BYTE 5
                    DB      ?       ;BYTE 6
                    DB      ?       ;BYTE 7
                    DB      ?       ;BYTE 8
                    DB      ?       ;BYTE 9
                    DB      ?       ;BYTE 10
```

This is the space allocation for the actual E²PROM Look-up table in the 2817A.

```
;
;
E2PROM          SEGMENT
;
;
E2PROM_TABLE        DB      ?       ;BYTE 1
                    DB      ?       ;BYTE 2
                    DB      ?       ;BYTE 3
                    DB      ?       ;BYTE 4
                    DB      ?       ;BYTE 5
                    DB      ?       ;BYTE 6
                    DB      ?       ;BYTE 7
                    DB      ?       ;BYTE 8
                    DB      ?       ;BYTE 9
                    DB      ?       ;BYTE 10
```

The 'DI' register is used in the "WRITE-E2-DATA" subroutine above to reference the E²PROM. The following ASSUME directive and MOV instructions would be needed to assign the proper addresses to the segment registers:

```
;
      ASSUME        ES:E2PROM,SS:STACK
;
        MOV         AX,0
        MOV         SS,AX
        MOV         AX,8000H
        MOV         ES,AX
;
;
```

8259A Initialization

The following is the code to initialize the 8259A Programmable Interrupt Controller for use in the 2817A/8088 Demo board.

```
;                       EQUATES
;
;
;
;
INT_CNTRLR_A0_LOW       EQU     0H
INT_CNTRLR_A0_HIGH      EQU     1H
;
ICW1            EQU         00010111B       ;EDGE TRIGGERED MODE
                                            ;ADDR INTERVAL OF 4
                                            ;SINGLE MODE
                                            ;ICW4 NEEDED
```

```
;
ICW2        EQU     00100000B       ;TYPE 20H, (VECTOR OF 20H
                                    ;WILL COME FROM 8259A: IT'S
                                    ;VALUE IS MULTIPLIED BY 4,
                                    ;SO THAT AN INTERRUPT 0 TO
                                    ;THE 8259A WILL RESULT IN THE
                                    ;VECTOR BEING FETCHED FROM
                                    ;LOCATION 80H)
;
ICW4        EQU     00000001B       ;NOT SPECIAL FULLY NESTED
                                    ;MODE
                                    ;NON-BUFFERED MODE
                                    ;NORMAL EOI
                                    ;8086 MODE
;
OCW1        EQU     11111110B       ;MASK OUT ALL BUT INTERRUPT 0
;
OCW2        EQU     00100000B       ;END OF INTERRUPT COMMAND
                                    ;NON-SPECIFIC EOI COMMAND
;
;


;
;           RAM MEMORY SPACE
;
;
;
STACK       SEGMENT AT 0                ;RAM AREA LOCATED FROM 0 TO 1FFFH
            DB      128 DUP(?)          ;RESERVED LOC'S


;
INTR_VECT   DW      ?                   ;E2PROM READY INTERRUPT SERVICE
                                        ;SUBROUTINE VECTOR - OFFSET.
;
            DW      ?                   ;SEGMENT VALUE
;
;
E2_BUSY     DB      ?                   ;SOFTWARE FLAG TO INDICATE THAT
                                        ;AN E2PROM BLOCK TRANSFER IS IN
                                        ;PROGRESS
;
;
            ORG     1FFFH
TOP_OF_STACK DW     ?
;
STACK       ENDS
;
;
;
;
```

```
;
;           CODE TO INITIALIZE THE 8259A
;
;
;
        OUT     INT_CNTRLR_A0_LOW, ICW1     ;1ST INITIALIZATION
                                            ;CONTROL WORD
;
        OUT     INT_CNTRLR_A0_HIGH, ICW2    ;2ND INITIALIZATION CONTROL
                                            ;WORD
;
        OUT     INT_CNTRLR_A0_HIGH, ICW4    ;LAST INIT. CONTROL WORD


        OUT     INT_CNTRLR_A0_HIGH, 0CW1    ;MASK OUT UNUSED INTR LINES
;
        MOV     AX, OFFSET INTR_SERV_ROUT
        MOV     INTR_VECT, AX               ;LOAD THE OFFSET VALUE OF THE
                                            ;INTERRUPT SERVICE ROUTINE INTO
                                            ;THE 1ST TWO BYTES OF THE
                                            ;INTERRUPT VECTOR DOUBLEWORD
;
        MOV     AX, CS
        MOV     INTR_VECT+2, AX             ;LOAD THE SEGMENT VALUE INTO THE
                                            ;2ND TWO BYTES OF THE INTERRUPT
                                            ;VECTOR DOUBLEWORD
```

APPENDIX B
2817A/8088 APPLICATIONS DEMO

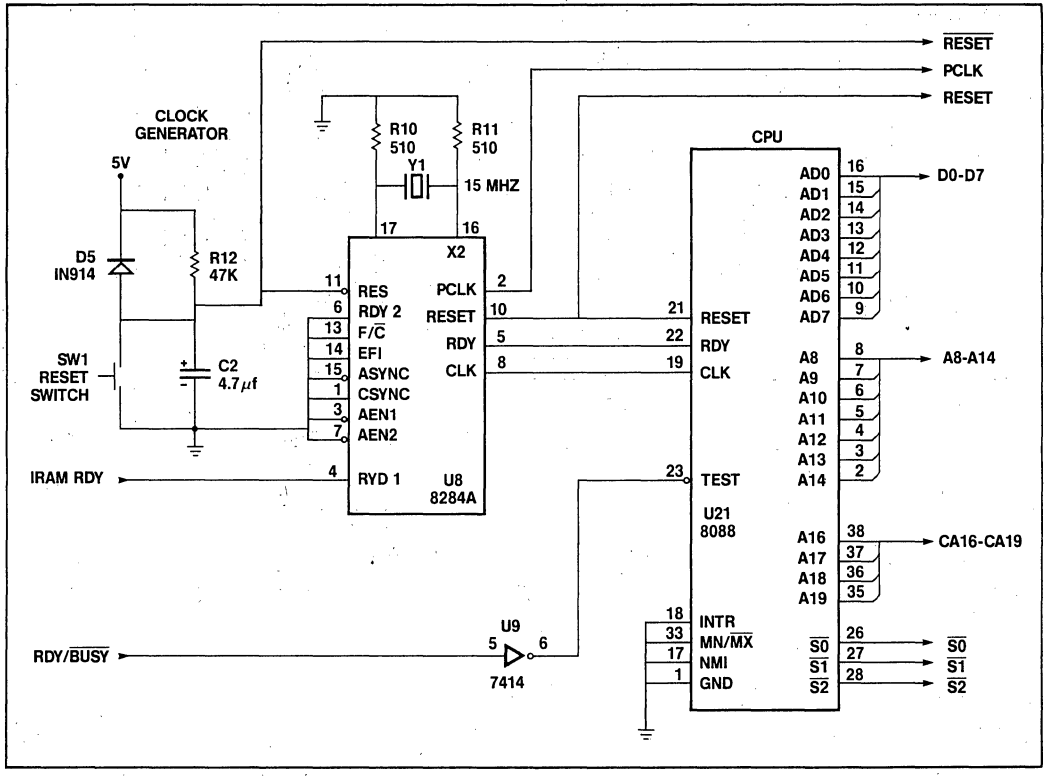


Figure 37. 2817A/8088 Application Demo

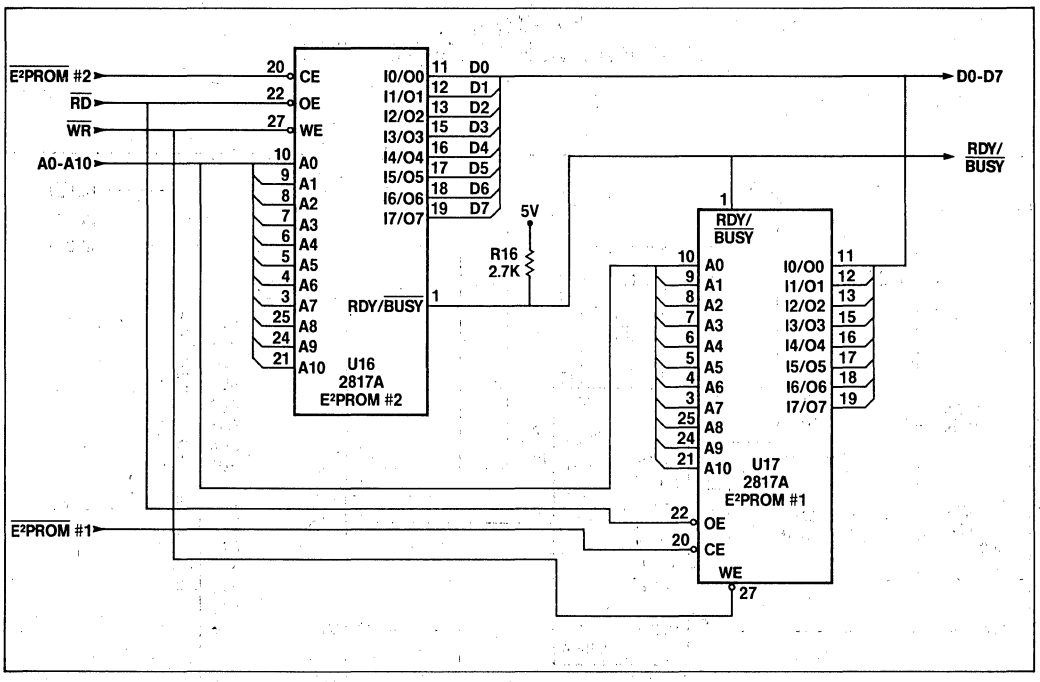


Figure 38. 2817A E²PROM Sockets

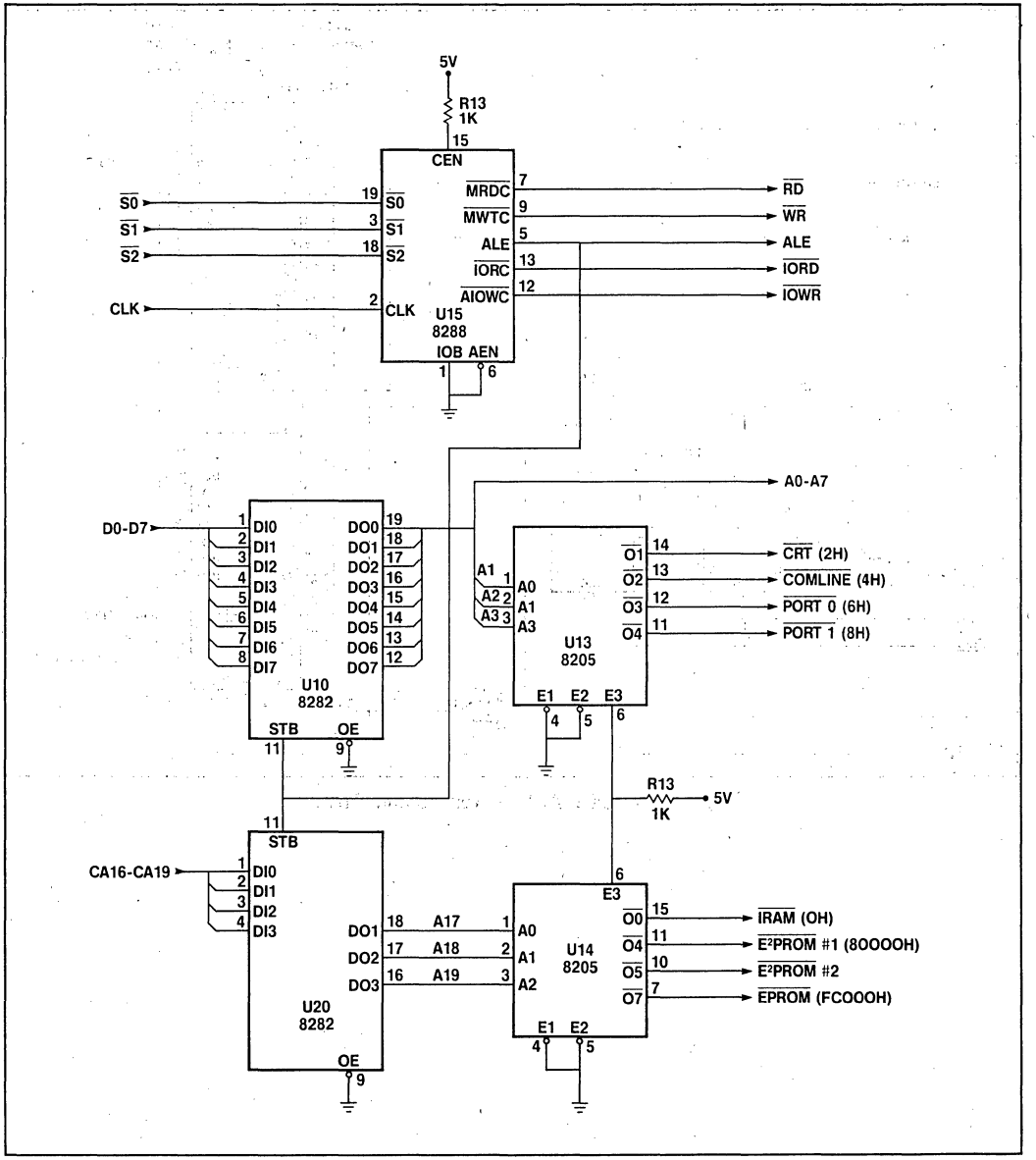


Figure 39. 2817A/8088 Applications Demo

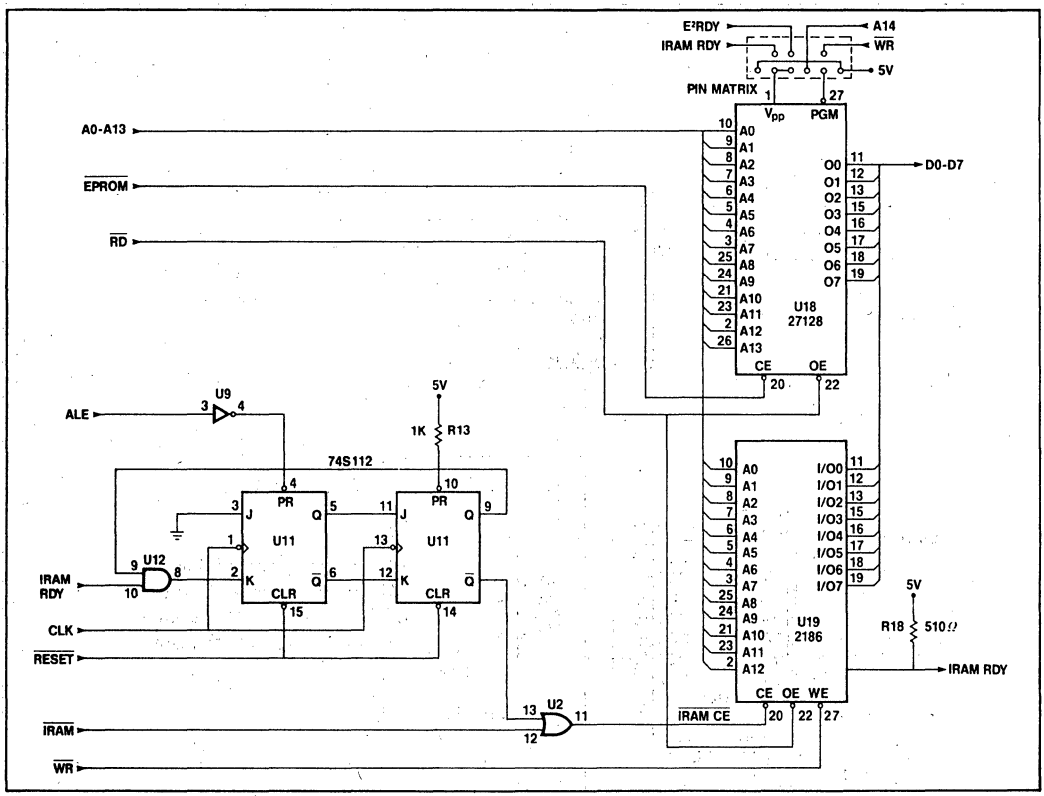


Figure 40. 2817A/8088 Applications Demo

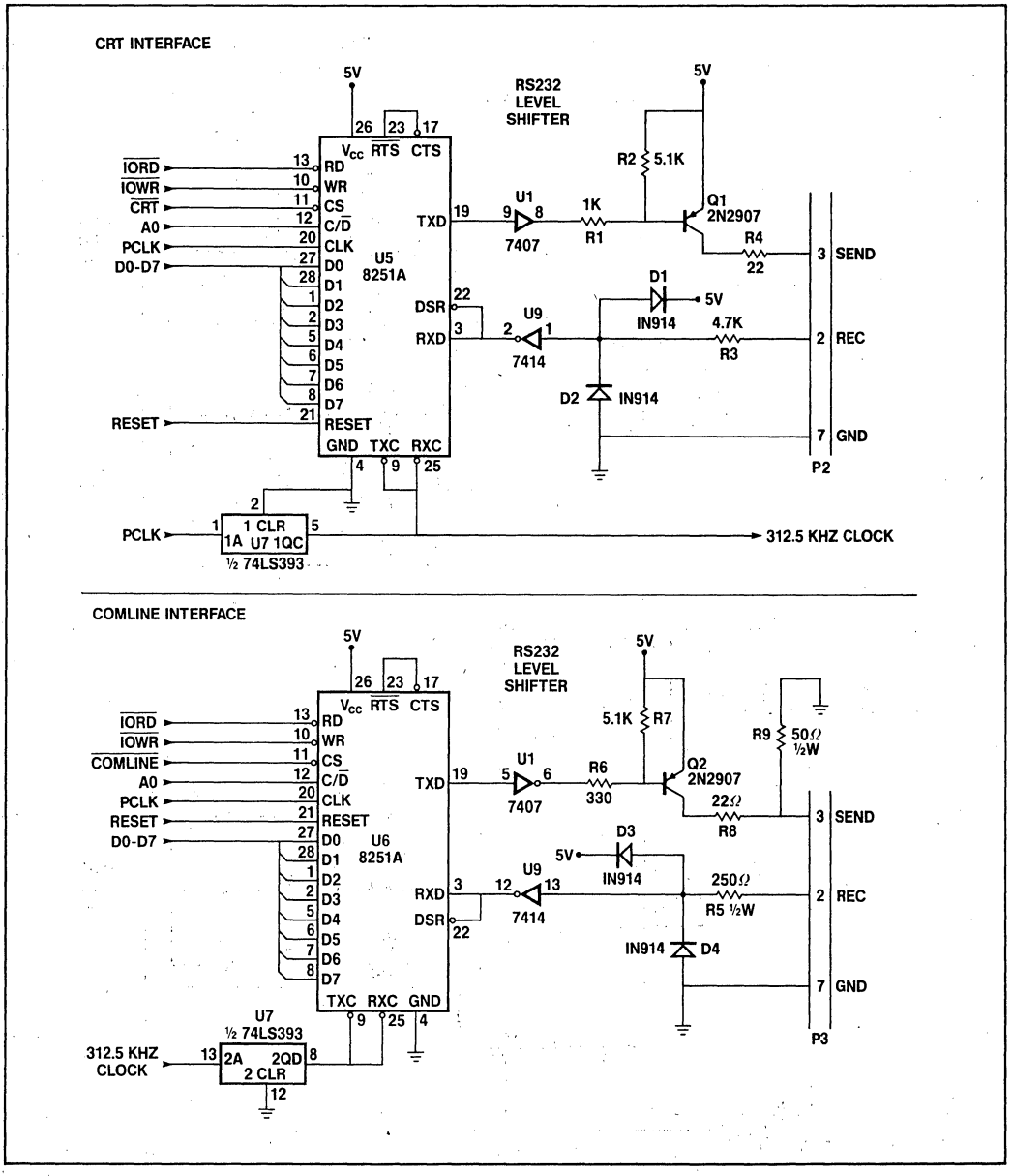


Figure 41. 2817A/8088 Applications Demo

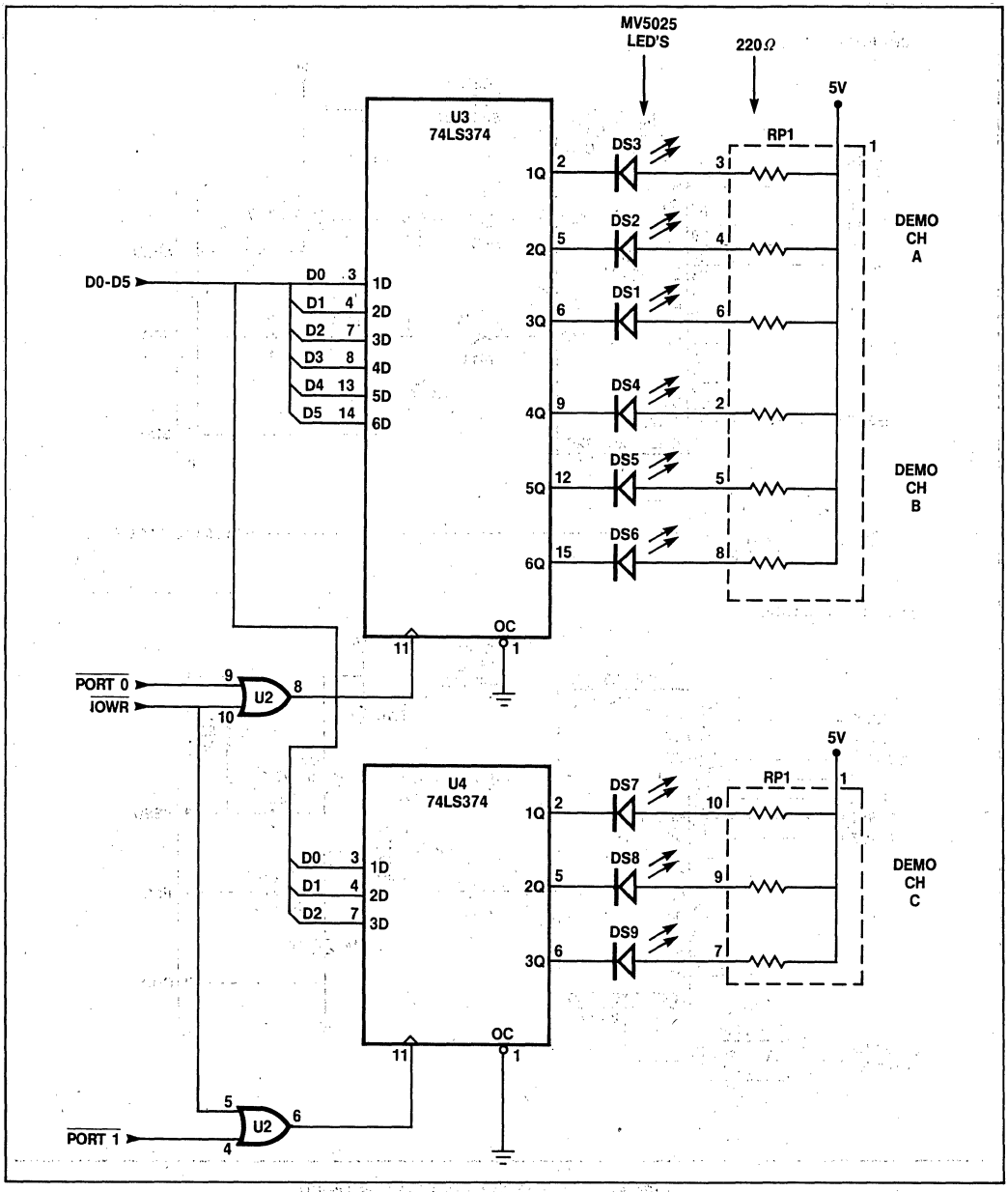
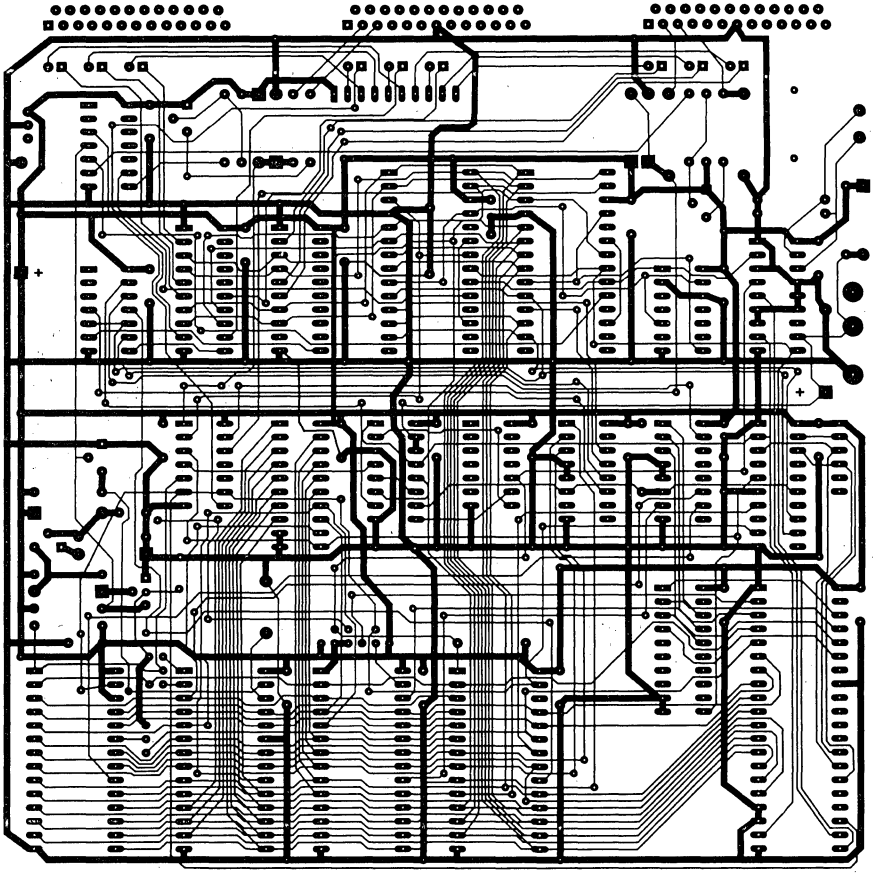


Figure 42. 2817A/8088 Applications Demo, Demo Channels A, B, C Display LED's

APPENDIX C
PRINTED CIRCUIT BOARD LAYOUT



APPENDIX D
PARTS LIST

2817A/8088 Applications Demo Parts List

IC's

Quantity	Description
1	D8088
1	D8284A
2	D8282
1	C2186-30
1	D27128-3
2	D3205
2	D8251A
1	74LS393
1	7407
1	74LS14
1	D8288
2	74LS374
1	74LS32
1	74S112
1	74LS08
1	2817A

Diodes

5	1N914
9	MV-5025

Capacitors

1	4.7 Microfarad
22	0.1 Microfarad Ceramic
1	25 Microfarad, 25V

Transistors

2	2N2907

Switches

1	Micro Switch #21SM284

Crystal

1	15 Megahertz

Sockets

4	28 Pin Low Profile Soldertail

Connectors

3	25 Pin RS232 Connector ITTJO-DBP-25SCA 8040
1	Molex Connector for Power Supply Housing #22-01-2037 2695 Series Peg #15-04-9210 Wafer #22-05-3031.7478 Series Terminals #08-56-0110 2759 Series

Headers

1	6 Pair Unit

Resistors

1	2.7K Ohm
1	4.7K Ohm
2	1K
3	510 Ohm
1	47K
2	5.1K
2	22 Ohm
1	330 Ohm
1	250 Ohm ½W
1	50 Ohm ½W
1	220 Ohm R-Pak 9 Pin SIP

APPENDIX E
FABRICATION/LAYOUT DRAWING